Eloísa Catarina Monteiro de Figueiredo Amaral e Macedo

Estudo prático de regularidade de problemas de Programação Semidefinida

Eloísa Catarina Monteiro de Figueiredo Amaral e Macedo

Estudo prático de regularidade de problemas de Programação Semidefinida

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, área de especialização Matemática Empresarial e Tecnológica, realizada sob a orientação científica da Dra. Tatiana Tchemisova Cordeiro, Professora Auxiliar do Departamento de Matemática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Jorge Sá Esteves
Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

vogais / examiners committee

Professora Doutora Maria do Carmo Miranda Guedes
Professora Auxiliar da Faculdade de Ciências da Universidade do Porto

Professora Doutora Tatiana Tchemisova Cordeiro
Professora Auxiliar do Departamento de Matemática da Universidade de Aveiro (orientadora)

agradecimentos / acknowledgements

Tive o prazer de ter como minha orientadora a Dra. Tatiana Tchemisova, a quem agradeço todo o apoio e toda a força que me prestou ao longo do desenvolvimento desta dissertação. Agradeço-lhe todas as ideias que enriqueceram este trabalho.

Merece também um agradecimento, o Dr. Jorge Sá Esteves pelo apoio aquando do início da minha aventura pelo MatLab para a implementação do algoritmo.

Agradeço ao Dr. Kortanek por se ter mostrado interessado e disponível em facultar-me informações e artigos que me poderiam ajudar neste projecto.

Agradeço à Elisabete, a amiga incentivadora, que aturou muitos dos meus picos de ansiedade; à Paula, por sempre acreditar que eu seria capaz e por todos os pequenos momentos de boa disposição.

Não me podia esquecer do Rui, companheiro nos bons e maus momentos e que ao longo do tempo em que tive de estar concentrada neste trabalho, me amparou lágrimas e me acarinhou nos dias em que me sentia esgotada. Obrigado por seres uma peça no puzzle da minha vida!

As últimas palavras de agradecimento são destinadas aos meus pais, Gabi e Albanito, os meus melhores amigos e a quem dedico este trabalho. Agradeço-lhes todas as oportunidades que me ofereceram, toda a paciência que precisaram ter durante este tempo de estudo e por me terem incutido o gosto pelo conhecimento e os valores que me ajudaram a ser uma melhor pessoa. Obrigado a todos!

Palavras-chave Programação Semi-Infinita (SIP), Programação Semidefinida (SDP), Subespaço de índices imóveis, Qualificações de Restrições (CQ - Constraint Qualifications), Condições de Optimalidade.

Resumo Um problema linear de Programação Semidefinida (SDP) consiste na minimização de uma função linear sujeita à condição de que a função matricial linear seja semidefinida.
Um problema de SDP considera-se regular se certas condições estão satisfeitas. Há diferentes caracterizações de regularidade de um problema, sendo uma delas a verificação da condição de Slater. Os problemas regulares de SDP têm sido estudados e as condições de optimalidade para estes problemas têm a forma de teoremas clássicos do tipo Karush-Kuhn-Tucker, e são facilmente verificadas. Na prática, é frequente encontrar problemas não regulares. O estudo destes problemas é bem mais complicado. Por isso, tem surgido o interesse em estudar e testar a regularidade dos problemas de SDP e deduzir condições de optimalidade e métodos de resolução dos problemas não regulares.
Em Kostyukova e Tchemisova [32] é proposto um algoritmo, chamado Algoritmo DIIS (Algorithm of Determination of the Immobile Index Subspace), que permite verificar se as restrições de um dado problema de SDP satisfazem a condição de Slater. A teoria que serve de base à construção deste algoritmo assenta nas noções de índices e subespaço de índices imóveis, originalmente usadas em Programação Semi-Infinita (SIP), e transpostas em [32] para SDP. Este algoritmo constrói uma matriz básica do subespaço de índices imóveis, caso a condição de Slater não seja verificada. A dimensão desta matriz caracteriza o grau de não regularidade do problema.
O objectivo deste trabalho é estudar o Algoritmo DIIS, implementá-lo e testá-lo usando vários problemas de teste de diferentes bases de dados de problemas de SDP. O Algoritmo DIIS foi implementado e executado a partir do MatLab e os testes numéricos efectuados permitiram concluir que o programa construído verifica com sucesso a maioria dos problemas teste. Além disso, o algoritmo permite caracterizar o grau de não regularidade dos problemas de SDP e pode ser usado para construção de algoritmos de resolução dos problemas de SDP não regulares.

Keywords Semi-Infinite Programming (SIP), Semidefinite Programming (SDP), Subspace of immobile indices, Constraint Qualifications (CQ), Optimality Conditions.

Abstract A linear problem of Semidefinite Programming (SDP) consists of minimizing a linear function subject to the constraint that the linear matrix function is semidefinite.
An SDP problem is considered regular if certain conditions are satisfied. There are several characterizations of a problem regularity, one of which is checking the Slater condition. The regular SDP problems have been studied and the optimality conditions for these problems have the form of Karush-Kuhn-Tucker type theorems, and are easily verified. In practice it is common to find problems that are not regular. The study of these problems is far more complicated. Therefore, there has been interest in studying and testing the regularity of the problems and deduct the SDP optimality conditions and methods for solving non-regular problems.
In Kostyukova e Tchemisova [32] is proposed an algorithm, called Algorithm DIIS (Algorithm of Determination of the Immobile Index Subspace), which allows to check if the constraints of a given SDP problem satisfy the Slater condition. The theory that underlies the construction of this algorithm is based on the notions of subspace of immobile indices and immobile indices properties, originally used in Semi-Infinite Programming (SIP), and implemented in [32] for SDP. This algorithm constructs a basic matrix of the subspace of immobile indices if the Slater condition is not verified. The size of this matrix characterizes the degree of non-regularity of the problem.
The purpose of this work is to study the DIIS algorithm, implement and test it using several test problems of different databases of SDP problems. The DIIS algorithm was implemented and executed from MatLab and numerical tests carried out showed that the program checks successfully the majority of test problems. Moreover, the algorithm allows to characterize the degree of non-regular problems of SDP and can be used to construct algorithms for solving non-regular SDP problems.

"Vive cada dia como se fosse o último...
Num deles hás-de acertar!"

– *Autor Desconhecido*

Conteúdo

Capítulo 1

Introdução

1.1 Objectivos do trabalho

Recentemente, a Programação Semidefinida (vamos usar a abreviatura SDP, do inglês Semidefinite Programming) tem despertado grande interesse devido à sua aplicabilidade em inúmeros problemas relacionados, por exemplo, com engenharia e controlo.

Nos últimos anos, diversos pesquisadores estudaram e desenvolveram métodos práticos para resolver problemas de SDP. Nas iterações destes métodos são verificadas uma ou outra condição de optimalidade. Para verificar estas condições é necessário saber se o problema dado é ou não é regular.

Existem diferentes definições de regularidade de problemas de SDP. Uma delas está ligada com a chamada condição de Slater. Em [32] foi descrito um algoritmo que permite não só verificar a regularidade de um problema de SDP, mas também encontrar o grau de não regularidade desse problema.

O objectivo deste trabalho é estudar o algoritmo proposto em [32], implementá-lo na forma de um programa em MatLab e testá-lo para diferentes problemas de SDP Linear, disponíveis na bibliografia e nas bases de dados acessíveis. A implementação deste algoritmo permitirá testar a regularidade de problemas de SDP e aplicar as condições de optimalidade adequadas.

1.2 Estado de Desenvolvimento do Conhecimento

A SDP é uma área da Optimização que estuda problemas de minimização de funções sujeitas a desigualdades matriciais lineares (LMI, em inglês Linear Matrix Inequalities). Os problemas de SDP surgiram como uma extensão da classe de problemas de Programação Linear (LP - Linear Programming, em inglês).

Os problemas de LP pretendem minimizar uma função linear sujeita a restrições lineares. Os modelos de LP aparecem, por exemplo, em problemas de fluxos de redes, planeamento de produção, problemas de risco de investimento.

Enquanto que a LP teve um grande e rápido crescimento durante os anos 50 e 60, devido à eficiência do Método Simplex de Dantzig, a SDP foi mais lenta a atrair a atenção. Um dos motivos que despoletou a investigação em SDP tem que ver com a não aplicabilidade do Método Simplex quando a região admissível não é poliédrica (ver Helmberg [21], Vieira [61]).

Bellman e Fan parecem ter sido os primeiros a formular um problema de SDP, em 1963 (ver [5]). Em vez de considerarem um problema de LP na forma vectorial, substituiram o vector de variáveis por uma matriz de variáveis. Para o problema resultante, embora equivalente à formulação geral, mas bastante mais complicado, foi formulado o problema dual, foram estabelecidos teoremas chave na dualidade e foi mostrado que a regularidade é uma condição importante em SDP (ver Todd [54]).

Nos últimos anos, o interesse em SDP tem vindo a crescer devido às várias aplicações de SDP. Uma das primeiras aplicações, de acordo com Vandenberghe e Boyd [59], foi em problemas de desigualdades matriciais lineares decorrentes de sistemas e controlo; também se encontraram aplicações em engenharia, incluindo a optimização estrutural e processamento de sinais; e nos últimos anos foi descoberto que SDP é uma ferramenta amplamente utilizada na relaxação de problemas de optimização combinatória $\mathcal{NP}-$difíceis. Algumas aplicações de SDP estão descritas em Wolkowicz et al. [67], que inclui capítulos com várias aplicações em Optimização Combinatória, Programação Quadrática Não Convexa, Optimização Não Convexa, Teoria e Sistemas de Controlo, Desenho Estrutural e problemas em Estatística.

No início dos anos 70, Donath e Hoffman [11] mostraram que o problema de particionamento de grafos pode ser encarado como um problema de SDP, considerando um problema associado de optimização de valores próprios.

Em 1979, Lovász [37] formulou um problema de SDP que fornecia um limite para a capacidade de Shannon num grafo e assim encontrou a capacidade do pentágono, resolvendo um problema matemático que até então era aberto. Nesta altura o método mais eficiente para resolver problemas de SDP era o método do elipsóide. Em 1988, Grotschel et al. [19] investigaram a fundo a aplicação deste método a problemas de Optimização Combinatória, usando-o para aproximar a solução tanto de relaxações de LP, como de SDP. Lovász e Schrijver [38] mostraram mais tarde como os problemas de SDP podiam fornecer boas relaxações de problemas de Programação Inteira binária, melhores do que as relaxações conseguidas a partir de LP (ver Todd [54]).

À medida que surgiam, tanto teoria eficiente, como algoritmos, entre os anos 80 e 90 dá-se a explosão da investigação na área da SDP. Assim, a SDP tornou-se numa área de interesse e de intensa investigação nos últimos tempos.

As contribuições de Nesterov e Nemirovski [43, 44] e de Alizadeh [1], extremamente importantes, mostraram que a nova geração de métodos de ponto interior, impulsionado primeiramente por Karmarkar para LP, podiam estender-se à SDP. Em particular, Nesterov e Nemirovski debruçaram-se sobre a resolução de problemas convexos não lineares utilizando métodos de ponto interior, através do desenvolvimento da teoria das funções de barreira auto-concordantes. Estes trabalhos, juntamente com o trabalho de Goemans e Williamson [17] onde foi demonstrado que a relaxação obtida pela formulação como SDP

produzia melhores aproximações para o problema combinatório de corte máximo, conduziram ao recente interesse na área de SDP.

Também surgiram diversos artigos de, por exemplo, Vandenberghe e Boyd [57, 58], de Klerk [27] e Klerk, Roos e Terlaky [28], que incidem sobre a teoria e aplicações de SDP, de Helmberg et al. [22], que descrevem um método de resolução dos problemas de SDP, e livros, nomeadamente, [21] de Helmberg e [67] editado por Wolkowicz, Saigal e Vandenberghe, que é um compêndio de 877 referências que englobam tanto teoria, como algoritmos e aplicações de SDP.

Em 2002, na sua tese de doutoramento, Krishnan [35] mostra como a LP está intimamente ligada com SDP, mostrando como facilmente se pode transportar muita da teoria de LP para SDP. Neste trabalho, Krishnan frisa que a teoria de dualidade em SDP é bem mais especial do que em LP. Em 2003, Krishnan e Mitchell [36] apresentaram um artigo sobre o problema dos métodos de ponto interior para SDP, no que respeita ao aumento do número de restrições. Este problema tem vindo a ser motivo de investigação e imenso estudo de outras aproximações. Neste artigo os autores mostram também que é possível tratar um problema de SDP como um problema de Programação Semi-Infinita (SIP - Semi-Infinite Programming, em inglês).

As condições de optimalidade clássicas (do tipo Karush-Kuhn-Tucker, usualmente denotadas por KKT), de acordo com Klerk [27], Nocedal e Wright [45], Pedregal [46] assumem que uma qualificação de restrições (CQ, Constraint Qualification) está satisfeita. Uma das condições comummente usada é a condição de Slater que consiste em admissibilidade estrita do problema. Na ausência de uma CQ, as condições clássicas de optimalidade podem falhar, não produzindo um resultado conclusivo (ver Tunçel e Wolkowicz [56]). Uma vez que nem todas as qualificações de restrições são fáceis de verificar, surge então o interesse em formular condições de optimalidade que não exijam condições adicionais. Tais condições foram propostas, por exemplo, em Jeyakumar e Nealon [26], Kostyukova e Tchemisova [32], Tunçel e Wolkowicz [56]. Em Kostyukova e Tchemisova [32], foram propostas condições de optimalidade para problemas lineares de SDP sem recurso à exigência de CQ. Na base destas condições está a noção de subespaço de índices imóveis e foi demonstrado que este subespaço é nulo se e só se o problema satisfaz a condição de Slater. Foi também proposto o algoritmo de construção de uma matriz básica deste subespaço, o Algoritmo DIIS.

1.3 Organização da Dissertação

Este trabalho está organizado em 5 capítulos e 3 apêndices. No primeiro capítulo é feita uma introdução à SDP.

No capítulo 2 é definido o problema que vamos estudar, um problema linear de SDP, apresentando-se conceitos cruciais à compreensão do nosso estudo e ainda as condições de optimalidade usuais; enumeram-se alguns métodos que são apropriados para a resolução de problemas de SDP, dando-se especial destaque ao método primal-dual de ponto interior, usado inicialmente para problemas de LP.

No capítulo 3, apresentamos noções novas de SDP e SIP, tais como as de índices imóveis e de subespaço de índices imóveis, que serão necessárias para a nova abordagem e que são objectos principais do estudo que apresentaremos. A seguir estudamos o algoritmo DIIS, que determina uma base do subespaço de índices imóveis, proposto em [32], e descrevemos os traços gerais da implementação deste algoritmo num programa em MatLab.

No capítulo 4 são apresentados os resultados da experiência computacional que validam o programa implementado e procede-se à sua análise e discussão, dando-se especial ênfase a como o programa pode ser usado para estudar a regularidade de problemas de SDP. Os resultados obtidos são comparados com os estudos apresentados em Freund, Ordóñez e Toh [12] e Jansson et al. [25].

Por fim, no capítulo 5 tecem-se algumas conclusões sobre o trabalho desenvolvido e são desenhadas as perspectivas de trabalho futuro.

No apêndice A apresentamos uma tabela com as principais características de *solvers* de problemas de SDP e no apêndice B explicamos como construir um ficheiro em formato SDPA esparso. Podem também ser encontrados os detalhes práticos da implementação do Algoritmo DIIS, bem como o código-fonte do algoritmo em linguagem MatLab. No apêndice C exemplificamos como transformar um problema de LP num problema de SDP.

Capítulo 2

Programação Semidefinida

Neste capítulo vamos apresentar alguns aspectos teóricos fundamentais da Programação Semidefinida. Primeiramente, vamos introduzir algumas definições básicas, assim como a notação que será utilizada ao longo do trabalho. Vamos introduzir as formulações do problema primal de SDP e do seu dual, bem como noções de dualidade fraca e forte e formulamos as condições de optimalidade clássicas para um problema linear de SDP.

Numa última parte deste capítulo, apresentamos alguns exemplos de aplicações de SDP e discutimos os métodos mais populares de resolução de problemas de SDP.

2.1 Notações e Definições Básicas

2.1.1 Conjuntos convexos e funções convexas

Definição 1 *Um conjunto $S \subset \mathbb{R}^n$ chama-se conjunto convexo se $\forall x, y \in S$ e $\forall \lambda \in [0,1]$ se tem*

$$\lambda x + (1-\lambda)\, y \in S.$$

Definição 2 *Seja $\{x_1, \ldots, x_p\}$ um conjunto de $p \geq 1$ pontos de $\mathbb{R}^n$ e $\{\lambda_1, \ldots, \lambda_p\}$ um conjunto de escalares. Se $\sum_{i=1}^{p} \lambda_i = 1$, então a $x = \sum_{i=1}^{p} \lambda_i x_i$ chama-se combinação afim dos pontos $x_1, \ldots, x_p$. Se adicionalmente $\lambda_i \geq 0$, $\forall i \in \{1, \ldots, p\}$, então a $x = \sum_{i=1}^{p} \lambda_i x_i$ chama-se combinação linear convexa dos pontos $x_1, \ldots, x_p$.*

A seguinte proposição pode ser considerada como uma alternativa à Definição 1:

Proposição 1 *Um conjunto $S \subset \mathbb{R}^n$ é convexo se, contendo quaisquer dois vectores $x \in S$ e $y \in S$, também contém todos os pontos do segmento de recta que os une, ou seja, contém também todas as suas combinações lineares convexas.*

Definição 3 *Seja $S \subset \mathbb{R}^n$ um conjunto convexo. A função $f : S \to \mathbb{R}$ diz-se convexa se $\forall x, y \in S$ e $\forall \lambda \in [0,1]$ se tem*

$$f\left(\lambda x + (1-\lambda)\right) \leq \lambda f\left(x\right) + (1-\lambda)\, f\left(y\right).$$

Definição 4 *O epigrafo de uma função $f : \mathbb{R}^n \to \mathbb{R}$ é o subconjunto de $\mathbb{R}^{n+1}$ definido por*

$$epi(f) = \{(x, r), x \in \mathbb{R}^n, r \in \mathbb{R} : \ r \geq f(x)\}.$$

A proposição seguinte estabelece a ligação entre funções convexas e conjuntos convexos.

Proposição 2 *Dada a função $f : S \subset \mathbb{R}^n \to \mathbb{R}$, com S um conjunto convexo, a função f é convexa se e só se o $epi(f)$ é um conjunto convexo.*

Exemplo 1 *Qualquer função linear é uma função convexa, uma vez que o epigrafo associado é um conjunto convexo (semiespaço).*

Exemplo 2 *Dados $b \in \mathbb{R}$ e $v \in \mathbb{R}^n \backslash \{0\}$, o semiespaço em $\mathbb{R}^n$ na forma*

$$\left\{x \in \mathbb{R}^n : v^T x \leq b\right\},$$

é um conjunto convexo.

Exemplo 3 *Dados $b \in \mathbb{R}$ e $v \in \mathbb{R}^n \backslash \{0\}$, o hiperplano $\left\{x \in \mathbb{R}^n : v^T x = b\right\}$ é convexo.*

Proposição 3 *A intersecção finita de conjuntos convexos é um conjunto convexo.*

Exemplo 4 *Um poliedro resulta da intersecção de um número finito de semiespaços (conjuntos convexos); logo é um conjunto convexo.*

Definição 5 *Um conjunto $K \subset \mathbb{R}^n$ diz-se um cone se, $\forall x \in K$ e $\forall \lambda \geq 0$ se tem $\lambda x \in K$.*

Adicionalmente, se o cone K é um conjunto convexo, então diz-se que K é cone convexo.

Dados $z, x \in \mathbb{R}^n$, denote-se por $\langle z, x \rangle$ o produto interno de z e x:

$$\langle z, x \rangle = \sum_{i=1}^{n} z_i x_i.$$

Definição 6 *Seja K um cone. Ao conjunto K^* na forma*

$$K^* = \{z : \langle z, x \rangle \geq 0, \forall x \in K\}$$

chamamos cone dual de K.

Dados inteiros $n \geq 1$ e $p \geq 1$, $\mathbb{R}^{n \times p}$ denota o conjunto de todas as $n \times p$ matrizes cujos elementos são números reais (matrizes reais).

A matriz $A \in \mathbb{R}^{n \times n}$ chama-se simétrica se $\forall i = 1, ..., n, \ j = 1, ..., n, \ a_{ij} = a_{ji}$.

O subespaço de todas as matrizes reais e simétricas de ordem n é designado aqui por

$$\mathcal{S}(n) = \left\{A \in \mathbb{R}^{n \times n} : A = A^T\right\} \subseteq \mathbb{R}^{n \times n}.$$

É fácil mostrar que $\mathcal{S}(n)$ é um conjunto convexo.

Definição 7 *Uma matriz $A \in \mathcal{S}(n)$ diz-se semidefinida (definida) positiva se uma das seguintes condições for válida:*

- $x^T Ax \geq 0 (> 0), \forall x \in \mathbb{R}^n (\forall x \in \mathbb{R}^n \backslash \{0\})$;
- *se os seus valores próprios forem todos não negativos (positivos).*

Proposição 4 *Considere-se uma matriz $A \in \mathbb{R}^{n\times n}$ com* $A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$.

A é uma matriz definida positiva se e só se todas as submatrizes principais da forma $A_1 = \begin{bmatrix} a_{11} \end{bmatrix}$, $A_2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $A_3 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$, $\dots A_n = A$ *têm determinantes positivos.*

A proposição anterior é usualmente chamada Critério de Sylvester.

Definição 8 *Seja $A \in \mathbb{R}^{n\times n}$, $k \in \{1, ..., n\}$ e $1 \leq i_1 < i_2 < ... < i_k \leq n$. O menor principal da matriz A de ordem k constituído pelas linhas e colunas com índices $i_1, ...i_k$ é definido mediante a fórmula:*

$$\delta_{i_1,...i_k}(A) := det(A_{\{i_1,...i_k\}\{i_1,...i_k\}}).$$

Proposição 5 *Uma matriz $A \in \mathbb{R}^{n\times n}$ é semidefinida positiva se e só se todos os menores principais são não negativos.*

Nota 1 *A matriz A é definida negativa se $-A$ é definida positiva. A matriz A é semidefinida negativa se $-A$ é semidefinida positiva.*

Exemplo 5 *Considere-se a matriz $A \in \mathcal{S}(3) \subset \mathbb{R}^{3\times 3}$ dada por* $A = \begin{bmatrix} -1 & 2 & 1 \\ 2 & -6 & -4 \\ 1 & -4 & -3 \end{bmatrix}$.

Trata-se de uma matriz semidefinida negativa, uma vez que a matriz $-A$ é semidefinida positiva, como se pode verificar através do cálculo de todos os menores principais da matriz $\begin{bmatrix} 1 & -2 & -1 \\ -2 & 6 & 4 \\ -1 & 4 & 3 \end{bmatrix}$:

$$\delta_1 = 1 \quad \delta_2 = 6 \quad \delta_3 = 3 \quad \delta_{\{1,2\}} = 10 \quad \delta_{\{1,3\}} = 2 \quad \delta_{\{2,3\}} = 2 \quad \delta_{\{1,2,3\}} = 0.$$

Dada uma matriz $A \in \mathcal{S}(n)$, escrevemos $A \succeq 0$ ($A \succ 0$) se A é semidefinida positiva (definda positiva), e $A \preceq 0$ ($A \prec 0$) se A é semidefinida negativa (definida negativa).

O conjunto das matrizes $n \times n$ simétricas semidefinidas positivas é designado por

$$\mathcal{P}(n) = \{A \in \mathcal{S}(n) : A \succeq 0\}.$$

Proposição 6 *O conjunto $\mathcal{P}(n)$ é um cone convexo.*

Prova: Para provar a proposição, temos de mostrar que

1. $\forall A \in \mathcal{P}(n), \forall \lambda \geq 0 \Rightarrow \lambda A \in \mathcal{P}(n)$
2. $\forall A, B \in \mathcal{P}(n), \forall \lambda \in [0,1] \Rightarrow \lambda A + (1-\lambda)B \in \mathcal{P}(n)$.

1. Tendo em conta que

   $A \in \mathcal{P}(n) \Leftrightarrow v^T Av \geq 0, \forall v \in \mathbb{R}^n \Leftrightarrow \lambda v^T Av \geq 0, \forall \lambda \geq 0 \Leftrightarrow v^T (\lambda A) v \geq 0,$
   logo $\lambda A \in \mathcal{P}(n)$ e temos que a condição 1 é verificada.

2. Seja agora $\lambda \in [0,1]$ e considerem-se duas matrizes $A, B \in \mathcal{P}(n)$. Para cada $v \in \mathbb{R}^n$ tem-se
   $$v^T(\lambda A + (1-\lambda)B)v = \lambda v^T Av + (1-\lambda)v^T Bv \geq 0$$
   o que significa que $\lambda A + (1-\lambda)B \in \mathcal{P}(n)$. ■

Definição 9 *O traço de uma matriz quadrada $A \in \mathbb{R}^{n\times n}$ é a soma dos elementos da sua diagonal principal:*

$$tr(A) = \sum_{i=1}^{n} a_{ii}.$$

Sendo $A, B \in \mathbb{R}^{n\times n}$, o traço de duas matrizes vem como

$$tr(AB) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{ji}.$$

A propriedade que se segue pode ser encontrada em Seber [52].

Propriedade 1 *Sendo $A, B, Z \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}$, podemos formular as seguintes propriedades:*

- $tr(A) = tr(A^T)$
- $tr(A+B) = tr(A) + tr(B)$
- $tr(cA) = ctr(A)$.

O espaço $\mathcal{S}(n)$ pode ser considerado como um espaço vectorial munido do produto interno definido pelo traço do produto de matrizes simétricas (ver Wolkowicz et al. [67]).

Uma propriedade importante do produto interno assim definido (ver Ramana [49]) aplicada ao cone $\mathcal{P}(n)$ é a seguinte:

Propriedade 2 *Sejam $A, B \in \mathcal{P}(n)$. Então, $tr(AB) \geq 0$ e a igualdade verifica-se se e só se $AB = 0_n$, onde $0_n \in \mathcal{S}(n)$ é a matriz nula.*

2.1.2 Problemas de Programação Convexa

Definição 10 *O problema geral de Programação Matemática (MP, do inglês, Mathematical Programming) é dado por*

$$\min f(x) \quad s.a\ x \in \mathcal{X}. \tag{2.1}$$

À função f chama-se função objectivo e ao conjunto $\mathcal{X}$ chama-se conjunto admissível.

Nota 2 *Quando o problema a considerar é de maximização podemos transformá-lo num problema de minimização, fazendo uma adequada mudança de sinal na função objectivo, sujeito às mesmas restrições. Ou seja,*

$$max \quad f(x) \quad s.a \quad x \in \mathcal{X} \Leftrightarrow min \ -f(x) \quad s.a \quad x \in \mathcal{X}.$$

Um problema de MP chama-se problema de Programação Não Linear se $\mathcal{X}$ é o conjunto definido pelas restrições

$$h_i(x) = 0, i \in I, \tag{2.2}$$
$$g_j(x) \leq 0, j \in J. \tag{2.3}$$

As igualdades (2.2) chamam-se restrições igualdades e as desigualdades (2.3) restrições desigualdades, sendo as funções h_i e g_j funções das respectivas restrições. As funções f, h_i, $i \in I$ e g_j, $j \in J$ são funções de variáveis reais e I e J são dois conjuntos finitos de índices.

Definição 11 *A solução admissível x^*, do problema de MP (2.1), é o minimizante (ou solução óptima) global se*

$$f(x^*) \leq f(x), \forall x \in \mathcal{X},$$

sendo $f(x^*)$ o mínimo de f em $\mathcal{X}$ (ou valor óptimo).

Definição 12 *A solução admissível x^0, do problema de MP (2.1), é um minimizante local se e só se existe uma vizinhança $V(x^0)$ de $\mathbb{R}^n$ de x^0, tal que x^0 é o minimizante global do problema*

$$\begin{aligned} &\min f(x) \\ s.a\ &h_i(x) = 0, i \in I \\ &g_j(x) \leq 0, j \in J \\ &x^0 \in V(x^0). \end{aligned}$$

Definição 13 *Dado um problema de MP na forma (2.1) e uma sua solução admissível $\bar{x} \in \mathcal{X}$, ao conjunto*

$$J_a(\bar{x}) = I \cup \{j \in J : g_j(\bar{x}) = 0\},$$

chama-se conjunto de índices activos, sendo as correspondentes restrições desigualdades designadas restrições activas em $\bar{x}$ e as restrições $g_j(\bar{x}) < 0$, restrições passivas em $\bar{x}$.

Definição 14 *O problema de MP na forma (2.1) diz-se convexo se $\mathcal{X}$ for um conjunto convexo em $\mathbb{R}^n$ e se $f : \mathbb{R}^n \to \mathbb{R}$ for uma função convexa.*

A Programação Convexa é uma área da MP que estuda problemas convexos.

Proposição 7 *Dado um problema convexo de MP, cada solução local é a sua solução global.*

Classes particulares de problemas convexos de Programação Matemática

Problemas de Programação Linear (LP)

Chama-se problema geral de LP a um problema na forma (2.1) onde $\mathcal{X}$ é definido pelas restrições (2.2) e (2.3) e onde as funções f, h_i, $i \in I$ e g_j, $j \in J$ são funções lineares.

É evidente que se trata de um problema convexo.

Problemas de Programação Semidefinida (SDP)

Um problema de programação semidefinida pode ser definido por

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{s.a } \mathcal{A}(x) \preceq 0, \end{aligned} \tag{2.4}$$

onde $\mathcal{A}(x) := \sum_{i=1}^{n} A_i x_i + A_0$, em que $A_i \in \mathcal{S}(s), i = 1, ..., n$ são matrizes reais simétricas.

O problema é convexo se a função f é convexa.

Problemas de Programação Não Linear (NLP, Nonlinear Programming)

É fácil verificar que um problema de NLP é convexo se as funções h_i são lineares e as funções f e g_j são convexas.

Problemas de Programação Quadrática

Um problema de programação quadrática pode ser definido por:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + d^T x \\ &\text{s.a } g_j(x) \leq 0, j \in J, \end{aligned} \tag{2.5}$$

onde Q é uma matriz $n \times n$ simétrica, d um vector de $\mathbb{R}^n$ e as funções g_j são lineares.

O problema é convexo se a função objectivo $f(x) = \frac{1}{2}x^T Qx + d^T x$ for convexa, e para isso, Q tem de ser semidefinida positiva.

Problemas de Programação Semi-Infinita (SIP, Semi-infinite Programming)

A designação programação semi-infinita surge quando um problema tem um número finito de variáveis sujeito a um número infinito de restrições ou um número infinito de variáveis sujeito a um número finito de restrições.

Assim, um problema de SIP pode ser descrito da seguinte forma:

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x) \\ &\text{s.a } g(x,t) \leq 0, \forall t \in T \in \mathbb{R}^s, \end{aligned} \tag{2.6}$$

onde T é o conjunto compacto de índices.

Este problema é convexo se a função f é convexa e as funções $g(x,t)$, com $t \in T$, são convexas em relação a x.

2.1.3 Condições de optimalidade para problemas de Programação Matemática

Dado um problema de MP e uma sua solução admissível, é importante saber verificar se esta solução é óptima. As condições de optimalidade (necessárias e suficientes) são as condições que permitem caracterizar a optimalidade de uma solução.

Na caracterização da optimalidade é atribuído um papel importante às chamadas qualificações de restrições (CQ). Estas são condições que se impõem nas restrições do problema de MP (finito ou infinito) para garantir que determinadas condições de optimalidade sejam satisfeitas. Se uma CQ não for satisfeita não podemos garantir que as condições de optimalidade são realmente necessárias.

A título de exemplo, enunciamos três CQ habitualmente usadas para problemas de NLP (ver Beck [4], Mangasarian [40], Nocedal e Wright [45]).

1. LICQ - Linear Independence Constraint Qualification (Independência Linear): os gradientes das restrições desigualdade activas e os gradientes das restrições igualdade em $x^* \in \mathcal{X}$ são linearmente independentes;

2. MFCQ - Mangasarian-Fromowitz Constraint Qualification: existe $w \in \mathbb{R}^n$ tal que $\bigtriangledown h_i(x^*)^T w = 0, \forall i \in I$ e $\bigtriangledown g_j(x^*)^T w < 0, \forall j \in J_a(x^*) \cap J$, e os gradientes das restrições igualdade são linearmente independentes em $x^* \in \mathcal{X}$;

3. Condição de Slater (ou condição de admissibilidade estrita): existe um ponto admissível $\bar{x} \in \mathcal{X}$ tal que todas as restrições desigualdade são estritamente satisfeitas em $\bar{x}$, ou seja, $g_j(\bar{x}) < 0, \forall j \in J$.

A condição de Slater é a mais forte das CQ's apresentadas. Facilmente se verifica que LICQ implica MFCQ, mas que o recíproco nem sempre é verdade (ver [45]).

A função de Lagrange é uma função importante para introdução de condições de optimalidade e que está na base da teoria de dualidade em Optimização.

Definição 15 *Dado um problema de NLP na forma*

$$\min_{x\in\mathbb{R}^n} f(x) \tag{2.7}$$
$$s.a\ h_i(x) = 0, i \in I \tag{2.8}$$
$$g_j(x) \leq 0, j \in J. \tag{2.9}$$

à função $\mathcal{L}(x,\lambda,\mu)$ *tal que* $x \in \mathbb{R}^n, \lambda \in \mathbb{R}^{|I|}, \mu \in \mathbb{R}_+^{|J|}$

$$\mathcal{L}(x,\lambda,\mu) := f(x) + \sum_{i\in I} \lambda_i h_h(x) + \sum_{j\in J} \mu_j g_j(x)$$

chama-se função de Lagrange ou Lagrangeano deste problema.

Aos coeficientes λ_i, com $i \in I$, e μ_j, com $j \in J$, chamam-se multiplicadores de Lagrange e ao vector (λ,μ), com $\lambda = (\lambda_i, i \in I)$ e $\mu = (\mu_j, j \in J)$, chama-se vector dos multiplicadores de Lagrange.

As condições de Karush-Kuhn-Tucker (KKT), introduzidas por Karush, Kuhn e Tucker, são conhecidas como as condições necessárias de optimalidade de I Ordem.

No teorema que se segue formulamos estas condições.

Teorema 1 *Seja* $x^* \in \mathcal{X}$ *um minimizante do problema (2.7)-(2.9) e assuma-se que uma das qualificações de restrições está satisfeita em* x^**. Neste caso, existe um vector de multiplicadores de Lagrange* $(\lambda^*,\mu^*) = \left(\lambda_i^*, i \in I; \mu_j^*, j \in J\right)$ *tal que as seguintes condições são satisfeitas em* (x^*,λ^*,μ^*) *:*

$$\nabla_x \mathcal{L}(x^*,\lambda^*,\mu^*) = 0, \tag{2.10}$$
$$h_i(x^*) = 0, \forall i \in I, \tag{2.11}$$
$$g_j(x^*) \leq 0, \forall j \in J, \tag{2.12}$$
$$\mu_j^* \geq 0, \forall j \in J, \tag{2.13}$$
$$\mu_j^* g_j(x^*) = 0, \forall j \in J. \tag{2.14}$$

A prova deste teorema pode ser encontrada, por exemplo, em Nocedal e Wright [45] (página 341).

As condições KKT são condições necessárias e não suficientes. Uma determinada solução admissível $x^* \in \mathcal{X}$ pode satisfazer as condições KKT e não ser minimizante do problema, por isso precisamos de condições de optimalidade mais fortes. Estas condições são condições necessárias de II Ordem e condições suficientes de optimalidade.

No entanto, se o problema (2.7)-(2.9) é convexo, então as condições KKT do Teorema 1 são também condições suficientes de optimalidade (ver Bazaraa [3], Helmberg [21], Nocedal e Wright [45]).

2.2 Problemas de Programação Semidefinida Linear

Um problema linear de SDP consiste na minimização ou maximização de uma função objectivo linear sujeita à condição de que a função matricial linear seja semidefinida e pode ser formulado na forma seguinte:

$$\begin{aligned} &\min c^T x \\ &\text{s.a } \mathcal{A}(x) \preceq 0, \end{aligned} \tag{2.15}$$

onde $c, x \in \mathbb{R}^n$ e a função matricial

$$\mathcal{A}(x) := \sum_{i=1}^{n} A_i x_i + A_0, \tag{2.16}$$

onde $A_i \in \mathcal{S}(s)$, com $i = 0, 1, ..., n$ e as matrizes $A_1, \ldots, A_n$ são linearmente independentes.

Existem várias formulações alternativas para um problema de SDP (ver Helmberg [21], Vandenberghe e Boyd [59], Wolkowicz et al. [67]); no entanto, todas são equivalentes à forma (2.15). Vamos obter uma delas usando a operação traço.

Introduzindo a variável de folga $S \in \mathcal{S}(s)$ em (2.15) obtemos

$$\begin{aligned} &\min c^T x \\ &\text{s.a } \sum_{i=1}^{n} A_i x_i + S = -A_0 \\ &S \succeq 0, \end{aligned} \tag{2.17}$$

onde $x \in \mathbb{R}^n$ e S são as variáveis.

Como as matrizes A_i, $i = 1, ..., n$, são linearmente independentes, o conjunto

$$\mathcal{V} = \left\{ S = -A_0 - \sum_{i=1}^{n} A_i x_i : x \in \mathbb{R}^n \right\}$$

é um subconjunto afim de $\mathcal{S}(s)$ de dimensão n. De acordo com Wolkowicz et al. [67], existem matrizes $G_j \in \mathcal{S}(s)$, com $j = 1, ..., k$, onde $k = \frac{1}{2}s(s+1) - n$, e um vector $g = (g_1, ..., g_k) \in \mathbb{R}^k$ tal que o conjunto $\mathcal{V}$ admite a representação equivalente

$$\mathcal{V} = \{S \in \mathcal{S}(s) : tr(G_j S) = g_j, j = 1, ..., k\}.$$

Pode-se determinar uma matriz $G_0 \in \mathcal{S}(s)$ que satisfaz

$$tr(G_0 A_i) = -c_i, i = 1, ..., n, \tag{2.18}$$

onde $c \in \mathbb{R}^n$, e tal que

$$c^T x = tr(G_0 S) + tr(G_0 A_0).$$

Esta igualdade é facilmente obtida, uma vez que

$$S = -A_0 - \sum_{i=1}^{n} A_i x_i \Leftrightarrow \sum_{i=1}^{n} A_i x_i = -A_0 - S$$

e tendo em conta (2.18),

$$c^T x = \sum_{i=1}^{n} c_i x_i = -tr\left(G_0 \sum_{i=1}^{n} A_i x_i\right) = tr(G_0 S) + tr(G_0 A_0).$$

Desta forma, podemos reformular o problema de SDP (2.15) como

$$\begin{array}{l} \min tr(G_0 S) \\ \text{s.a } tr(G_j S) = g_j, j = 1, ..., k \\ S \succeq 0. \end{array} \tag{2.19}$$

Exemplo 6 *Considere-se o problema de SDP:*

$$\begin{array}{c} \min x_{12} \\ s.a \begin{bmatrix} 0 & x_{12} & 0 \\ x_{12} & x_{22} & 0 \\ 0 & 0 & 1 + x_{12} \end{bmatrix} \succeq 0 \end{array} \tag{2.20}$$

ou, equivalentemente,

$$\begin{array}{c} \min \ x_{12} \\ s.a \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} x_{12} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} x_{22} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \preceq 0. \end{array}$$

Note-se que este problema é um problema na forma (2.15), com $c = (1, 0)^T$ *e*

$$A_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, A_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \text{ e } A_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Seja $S = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{12} & x_{22} & x_{23} \\ x_{13} & x_{23} & x_{33} \end{bmatrix}$ *uma matriz simétrica* 3×3.

Vamos mostrar que existem matrizes $G_j \in \mathcal{S}(3), j = 1, ...4$ *e um vector* $g \in \mathbb{R}^4$ *tal que*

$$tr\left(G_j S\right) = g_j, j = 1, ..., 4. \tag{2.21}$$

Repare-se que estamos à procura de umas quaisquer matrizes simétricas 3×3 *e de um vector de dimensão* 4 *que satisfaçam as condições (2.21).*

De facto, se escolhermos as matrizes

$$G_1 = \begin{bmatrix} 0 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, G_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, G_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, G_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

e o vector

$$g = \left[1\ 0\ 0\ 0\right]^T,$$

substituindo-os nas condições (2.21), obtemos

$tr\left(G_1 S\right) = g_1 \Leftrightarrow -x_{12} + x_{33} = 1 \Leftrightarrow x_{33} = 1 + x_{12}$

$tr\left(G_2 S\right) = g_2 \Leftrightarrow x_{11} = 0$

$tr\left(G_3 S\right) = g_3 \Leftrightarrow 2x_{13} = 0 \Leftrightarrow x_{13} = 0$

$tr\left(G_4 S\right) = g_4 \Leftrightarrow 2x_{23} = 0 \Leftrightarrow x_{23} = 0$

e concluímos que a matriz de restrições do problema (2.20) satisfaz estas condições.

Vamos agora determinar a matriz $G_0 \in \mathcal{S}(3)$ *na forma* $G_0 = \begin{bmatrix} g_{11}^0 & g_{12}^0 & g_{13}^0 \\ g_{12}^0 & g_{22}^0 & g_{23}^0 \\ g_{13}^0 & g_{23}^0 & g_{33}^0 \end{bmatrix}$ *e tal que*

$$tr\left(G_0 A_i\right) = -c_i, i = 1, 2 \tag{2.22}$$

$$c^T x = tr\left(G_0 S\right) + tr\left(G_0 A_0\right). \tag{2.23}$$

Então, por (2.22), sendo $c_1 = 1$ *e* $c_2 = 0$, *temos*

$tr\left(G_0 A_1\right) = -1 \Leftrightarrow -2g_{12}^0 - g_{33}^0 = -1 \Leftrightarrow g_{12}^0 = \frac{1 - g_{33}^0}{2}$

$tr\left(G_0A_2\right) = 0 \Leftrightarrow g_{22}^0 = 0.$

Para verificarmos a condição (2.23), precisamos de determinar $tr\left(G_0A_0\right)$ *e* $tr\left(G_0S\right)$.

$tr\left(G_0A_0\right) = -g_{33}^0$

$$tr\left(G_0S\right) = \left(g_{11}^0x_{11} + \tfrac{1-g_{33}^0}{2}x_{12} + g_{13}^0x_{13}\right) + \left(\tfrac{1-g_{33}^0}{2}x_{12} + 0 + g_{23}^0x_{23}\right) + \\ + \left(g_{13}^0x_{13} + g_{23}^0x_{23} + g_{33}^0x_{33}\right)$$

Assim,

$$c^Tx = tr\left(G_0S\right) + tr\left(G_0A_0\right)$$

$$\Leftrightarrow x_{12} = \left(g_{11}^0x_{11} + \tfrac{1-g_{33}^0}{2}x_{12} + g_{13}^0x_{13}\right) + \left(\tfrac{1-g_{33}^0}{2}x_{12} + 0 + g_{23}^0x_{23}\right) + \\ + \left(g_{13}^0x_{13} + g_{23}^0x_{23} + g_{33}^0x_{33}\right) - g_{33}^0$$

$$\Rightarrow \begin{cases} g_{11}^0 = 0 \\ g_{13}^0 = 0 \\ g_{23}^0 = 0 \\ g_{33}^0 = 0 \end{cases}$$

Portanto, $g_{12}^0 = \frac{1}{2}$ *e* $G_0 = \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

Desta forma obtemos

$$\begin{aligned} \min \quad & tr\left(\begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} S\right) \\ s.a \quad & tr\left(\begin{bmatrix} 0 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} S\right) = 1 \\ & tr\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} S\right) = 0 \\ & tr\left(\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} S\right) = 0 \\ & tr\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} S\right) = 0 \\ & S \succeq 0 \end{aligned}$$

onde S é uma matriz simétrica 3×3.

Notemos que um problema de Programação Linear pode ser considerado como um caso particular de SDP; para isso temos de supôr que as matrizes A_i são diagonais.

Dado um vector $v \in \mathbb{R}^n$, designamos por $diag(v)$ a matriz $n \times n$ cujos elementos da diagonal são as componentes do vector v e os restantes elementos são nulos.

Consideremos o problema de LP na forma

$$\begin{array}{c} \min c^T x \\ \text{s.a } Bx + b \leq 0 \end{array} \tag{2.24}$$

onde $B \in \mathbb{R}^{s \times n}, b, c, x \in \mathbb{R}^n$ e a desigualdade significa desigualdade componente a componente.

É evidente que, dado um vector $v \in \mathbb{R}^n$, $v \leq 0$ se e só se a matriz $diag(v)$ é semidefinida negativa.

Então, podemos reformular o problema (2.24) na forma (2.15), se suposermos

$$A_0 = diag(b) \text{ e } A_i = diag(B^i), i = 1, ..., n,$$

onde $B^i \in \mathbb{R}^s$, com $i = 1, ..., n$, é a $i-$ésima coluna (vector) da matriz B.

Exemplo 7 *Dado o problema de LP:*

$$\begin{array}{rl} \min & -x_1 + x_2 \\ s.a & x_1 - x_2 \geq 1 \\ & 2x_1 - x_2 \leq -4 \\ & -x_1 + 2x_2 \leq 2, \end{array}$$

a formulação equivalente como problema de SDP é:

$$\begin{array}{rl} \min & -x_1 + x_2 \\ s.a & \mathcal{A}(x) \preceq 0 \end{array}$$

onde $\mathcal{A}(x) := \sum_{i=1}^{2} A_i x_i + A_0$, *com*

$$A_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -2 \end{bmatrix}, A_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \text{ e } A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

Ao longo deste trabalho vamos chamar ao problema (2.15) problema primal de SDP.

Vamos mostrar que um problema de SDP é um problema convexo.
A função f é linear, logo convexa e para mostrar que o conjunto admissível do problema $\mathcal{X} = \{x \in \mathbb{R}^n : \mathcal{A} \preceq 0\}$ é convexo, vamos mostrar que $\mathcal{A}(\lambda x + (1-\lambda)y) \preceq 0$ para todo o $x \in \mathcal{X}$ e $y \in \mathcal{X}$ e todo o $\lambda \in [0,1]$.

De facto,

$$\begin{aligned}\mathcal{A}(\lambda x + (1-\lambda)y) &= A_0 + \sum_{i=1}^{n} A_i(\lambda x_i + (1-\lambda)y_i) \\ &= A_0 + \lambda \sum_{i=1}^{n} A_i x_i + (1-\lambda)\sum_{i=1}^{n} A_i y_i \\ &= \lambda A_0 + (1-\lambda)A_0 + \lambda \sum_{i=1}^{n} A_i x_i + (1-\lambda)\sum_{i=1}^{n} A_i y_i \\ &= \lambda \mathcal{A}(x) + (1-\lambda)\mathcal{A}(y) \preceq 0, \text{ uma vez que } 1-\lambda \geq 0.\end{aligned}$$

Assim, podemos concluir que o problema (2.15) é convexo.

2.3 Dualidade em Programação Semidefinida

A dualidade é uma ferramenta poderosa e amplamente utilizada em Optimização por uma série de razões (Klerk [27]).
Seguidamente, vamos considerar algumas noções de dualidade aplicadas a SDP.

Dado o problema primal de SDP na forma (2.15), a função de Lagrange associada ao problema é a função

$$\mathcal{L}(x, Z) = c^T x + tr\left(Z\mathcal{A}(x)\right) = c^T x + tr\left(Z\left(A_0 + \sum_{i=1}^{n} A_i x_i\right)\right),$$

com $Z \in \mathcal{P}(s)$ e $x \in \mathbb{R}^n$.
À matriz Z chama-se variável dual associada ao problema primal (2.15).
Usando as propriedades da função $tr(.)$, a função de Lagrange pode ainda ser escrita da seguinte forma:

$$\begin{aligned}\mathcal{L}(x, Z) &= c^T x + tr\left(Z\left(A_0 + \sum_{i=1}^{n} A_i x_i\right)\right) \\ &= \sum_{i=1}^{n} c_i x_i + tr\left(Z\left(A_0 + \sum_{i=1}^{n} A_i x_i\right)\right) \\ &= tr\left(ZA_0\right) + \sum_{i=1}^{n}\left(tr\left(ZA_i\right) + c_i\right)x_i, \text{ com } Z \in \mathcal{P}(s) \text{ e } x \in \mathbb{R}^n.\end{aligned}$$

Chamamos função primal ([61]) associada ao problema (2.15) à função

$$L_p(x) = \max_{Z \in \mathcal{P}(s)} \mathcal{L}(x, Z)$$

e a função dual associada será

$$\begin{aligned} L_d(Z) &= \min_{x \in \mathbb{R}^n} \mathcal{L}(x, Z) \\ &= \min_{x \in \mathbb{R}^n} c^T x + tr\left(Z\left(A_0 + \sum_{i=1}^{n} A_i x_i\right)\right) \\ &= \min_{x \in \mathbb{R}^n} tr\left(ZA_0\right) + \sum_{i=1}^{n} \left(tr\left(ZA_i\right) + c_i\right) x_i. \end{aligned}$$

Definição 16 *Chama-se problema primal associado a (2.15), ao problema*

$$\min_{x \in \mathbb{R}^n} L_p(x) = \min_{x \in \mathbb{R}^n} \left(\max_{Z \in \mathcal{P}(s)} \mathcal{L}(x, Z)\right), \tag{2.25}$$

e chama-se problema dual associado a (2.15), ao problema

$$\max_{Z \in \mathcal{P}(s)} L_d(Z) = \max_{Z \in \mathcal{P}(s)} \left(\min_{x \in \mathbb{R}^n} \mathcal{L}(x, Z)\right). \tag{2.26}$$

O problema dual (2.26) pode ser transformado na forma:

$$\begin{aligned} &\max \ tr\left(A_0 Z\right) \\ &\text{s.a} \ -tr\left(A_i Z\right) = c_i, \forall i = 1, \ldots, n \\ &Z \succeq 0. \end{aligned} \tag{2.27}$$

Note-se que (2.27) tem a forma (2.19), que por sua vez é equivalente a um problema de SDP na forma (2.15).

Nota 3 *A dualidade é simétrica, no sentido em que o dual do dual é o primal (ver Wolkowicz et al. [67]).*

Exemplo 8 *Consideremos o problema representado no Exemplo 6, (2.20).*

De acordo com a Definição 16 e como (2.26) pode ser transformado em (2.27), o dual do problema (2.20) tem a forma:

$$\begin{aligned} &\max \quad tr\left(A_0 Z\right) \\ s.a \quad &-tr\left(A_1 Z\right) = c_1 \\ &-tr\left(A_2 Z\right) = c_2 \\ &Z \succeq 0 \end{aligned}$$

onde $Z \in \mathbb{R}^{3\times 3}$ *é a variável dual,* $c_1 = 1$, $c_2 = 0$ *e as matrizes* A_0, A_1 *e* A_2 *são dadas por*

$$A_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{2.28}$$

Tendo em consideração (2.28), o problema dual ao problema (2.20) pode ser escrito na forma

$$\begin{aligned} \max \quad & tr\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} Z\right) \\ s.a \quad & -tr\left(\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} Z\right) = 1 \\ & -tr\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} Z\right) = 0 \\ & Z \succeq 0. \end{aligned}$$

Notemos que, uma vez que $Z = [z_{ij}] \in \mathcal{S}(3)$ *é uma matriz simétrica, o último problema pode ser reescrito na forma mais simples*

$$\begin{aligned} \max \quad & z_{33} \\ s.a \quad & -2z_{12} - z_{33} = 1 \\ & z_{22} = 0 \\ & z_{11},\ z_{13},\ z_{23},\ z_{33} \geq 0, \end{aligned}$$

que é um problema de LP, e que pode ser transformado num problema de SDP na forma (2.15) (ver Exemplo 7).

Exemplo 9 *Consideremos o problema de SDP:*

$$\begin{aligned} \min \quad & -tr\left(\begin{bmatrix} 4 & -1 \\ -1 & 5 \end{bmatrix} Z\right) \\ s.a \quad & tr\left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} Z\right) = 1 \\ & tr\left(\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} Z\right) = 1 \\ & Z \succeq 0, \end{aligned} \tag{2.29}$$

com $Z \in \mathbb{R}^{2\times 2}$.

Fazendo uma adequada mudança de sinal na função objectivo e nas restrições, este problema é da forma (2.27), ou seja, é o problema dual do problema:

$$\begin{array}{ll} \max & x_1 + x_2 \\ s.a & \mathcal{A}(x) \preceq 0 \end{array}$$

com

$$\mathcal{A}(x) = \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} x_2 + \begin{bmatrix} -4 & 1 \\ 1 & -5 \end{bmatrix}.$$

A teoria da dualidade em SDP tem muito em comum com a de LP, no entanto é mais especial (ver Klerk [27], Krishnan [35]). Tal como acontece em LP, a propriedade que vamos apresentar também se verifica em SDP, para qualquer par de soluções primal-dual admissíveis.

Propriedade 3 *(Dualidade Fraca) Sejam* $x \in \mathbb{R}^n$ *uma solução admissível do problema primal (2.15) e* $Z \in \mathcal{P}(s)$ *uma solução admissível do problema dual (2.27). Neste caso*

$$c^T x \geq tr\left(A_0 Z\right).$$

Prova: De facto, sendo $p = c^T x$ e $d = tr\left(A_0 Z\right)$, o valor $p - d$ vem como

$$\begin{aligned} p - d = c^T x - (tr\left(ZA_0\right)) \\ =& \sum_{i=1}^{n} c_i x_i - tr\left(ZA_0\right) \\ =& \sum_{i=1}^{n} \left(-tr\left(ZA_i\right)\right) x_i - tr\left(ZA_0\right), \text{ pela formulação do dual, } -tr\left(ZA_i\right) = c_i, \forall i \\ =& -tr\left(\left(A_0 + \sum_{i=1}^{n} A_i x_i\right) Z\right) \\ =& tr\left(\left(-A_0 - \sum_{i=1}^{n} A_i x_i\right) Z\right). \end{aligned}$$

A desigualdade resulta do facto de, tendo em conta que $-A_0 - \sum_{i=1}^{n} A_i x_i \succeq 0$ e que $Z \succeq 0$,

$$tr\left(\left(-A_0 - \sum_{i=1}^{n} A_i x_i\right) Z\right) \geq 0.$$

Assim, obtemos $p \geq d$, para quaisquer x, solução admissível do problema primal, e Z, solução admissível do dual.

Desta forma, a propriedade fica provada. ■

Tendo em consideração a propriedade de dualidade fraca, qualquer solução do problema dual induz um limite inferior para o valor óptimo da função objectivo do problema primal (ver Wolkowicz [64]).

Vamos introduzir a definição de desvio de dualidade (em inglês, *"duality gap"*).

Definição 17 *Denotemos por $p^* = c^T x^*$ o valor óptimo (caso exista) da função objectivo do problema primal (2.15) e por $d^* = tr(A_0 Z^*)$ o valor óptimo (caso exista) da função objectivo do seu dual (2.27). À diferença*

$$p^* - d^* \tag{2.30}$$

chama-se desvio de dualidade.

Tendo em consideração a propriedade de dualidade fraca, somos levados a concluir que o desvio de dualidade (2.30) é não negativo, ou seja, sempre está satisfeito $p^* - d^* \geq 0$.

Da teoria de LP é sabido que, se o problema (primal ou dual) tiver solução óptima, então o seu problema dual também tem solução óptima e o desvio de dualidade é nulo (ver, por exemplo, Kolman e Beck [30]). Esta propriedade não é válida para problemas de SDP. Para ilustrar esta situação, vamos considerar os seguintes exemplos.

Exemplo 10 *Consideremos o problema primal de SDP*

$$\min \; x_1$$

$$s.a \quad \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} x_2 + \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \preceq 0$$

e o seu dual

$$\max \quad tr\left(\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} Z \right)$$

$$s.a \quad -tr\left(\begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} Z \right) = 1$$

$$-tr\left(\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} Z \right) = 0$$

$$Z = \begin{bmatrix} z_1 & z_2 \\ z_2 & z_3 \end{bmatrix} \succeq 0$$

É fácil verificar que a solução óptima do problema primal tem de satisfazer as condições

$$x_1 \geq 0 \ e \ x_1 x_2 - 1 \geq 0.$$

Assim, $x_1^* = 1 \ e \ x_2^* \geq 1 \ e \ x^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Em relação ao problema dual, a sua solução óptima tem de satisfazer as seguintes condições:

$$z_1 = 1, z_3 = 0 \ e \ z_1 z_3 - {z_2}^2 \geq 0.$$

Então, $Z^* = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$.

Dadas as soluções óptimas dos problemas primal e dual, obtemos $p^* = 1$ *e* $d^* = 0$. *Neste caso, o desvio de dualidade* $p^* - d^* = 1$ *e não é nulo.*

Exemplo 11 *Consideremos agora um outro problema de SDP*

$$\begin{array}{c} \min \ x_1 \\ s.a \quad \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} x_2 \preceq 0 \end{array}$$

e o seu dual

$$\begin{array}{rc} & \max \quad 0 \\ s.a & -tr\left(\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} Z\right) = 1 \\ & -tr\left(\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} Z\right) = 0 \\ & Z = \begin{bmatrix} z_1 & z_2 \\ z_2 & z_3 \end{bmatrix} \succeq 0. \end{array}$$

A solução óptima do primal tem de verificar as condições:

$$-{x_1}^2 \geq 0 \ e \ x_2 \in \mathbb{R}.$$

A desigualdade só se verifica se $x_1^* = 0$. *Então,* $p^* = 0$.

Quanto à solução do problema dual, esta tem de satisfazer as seguintes condições:

$$z_2 = \frac{1}{2}, z_3 = 0, z_1 \geq 0 \text{ e } z_1 z_3 - {z_2}^2 \geq 0.$$

É evidente que estas condições são incompatíveis, uma vez que a última condição não é verificada para $z_2 = \frac{1}{2}$. Concluímos que o problema dual não admite solução admissível.

Atentemos agora num exemplo em que o desvio de dualidade é nulo.

Exemplo 12 *Consideremos o problema de SDP:*

$$\begin{array}{ll} & \min \ x_1 \\ s.a & \left[\begin{array}{cc} -1 & 0 \\ 0 & 0 \end{array}\right] x_1 + \left[\begin{array}{cc} 0 & 0 \\ 0 & 1 \end{array}\right] x_2 \preceq 0 \end{array}$$

e o seu dual

$$\begin{array}{ll} & \max \ 0 \\ s.a & -tr\left(\left[\begin{array}{cc} -1 & 0 \\ 0 & 0 \end{array}\right] Z\right) = 1 \\ & -tr\left(\left[\begin{array}{cc} 0 & 0 \\ 0 & 1 \end{array}\right] Z\right) = 0 \\ & Z = \left[\begin{array}{cc} z_1 & z_2 \\ z_2 & z_3 \end{array}\right] \succeq 0. \end{array}$$

A solução óptima do problema primal é $x^* = \left[\begin{array}{c} 0 \\ 0 \end{array}\right]$ e para o dual, a solução óptima é $Z^* = \left[\begin{array}{cc} 1 & 0 \\ 0 & 0 \end{array}\right]$.

Obtemos $p^* = d^* = 0$ e a propriedade de dualidade forte é satisfeita.

Definição 18 *Dado o problema de SDP (2.15), dizemos que a condição de Slater está satisfeita se*

$$\exists \bar{x} \in \mathbb{R}^n : \mathcal{A}(\bar{x}) \prec 0.$$

O teorema que se segue permite-nos concluir que, se a condição de Slater está satisafeita no problema primal de SDP, então é satisfeita a dualidade forte.

Teorema 2 *(Dualidade Forte) Suponhamos que as restrições do problema (2.15) satisfazem a condição de Slater. Então o problema (2.15) e o seu dual na forma (2.27) admitem soluções óptimas e o desvio de dualidade é nulo, ou seja, $p^* = d^*$.*

A demonstração deste resultado pode ser consultada em Vandenberghe e Boyd [59] (página 18).

2.4 Condições de Optimalidade para Problemas de SDP

Baseado em Klerk [27], Wolkowicz [65], vamos apresentar um teorema que permite caracterizar as soluções de um problema de SDP como óptimas.

Vamos considerar o problema primal de SDP na forma (2.17):

$$\begin{aligned} & \min c^T x \\ \text{s.a } & \sum_{i=1}^{n} A_i x_i + S = -A_0 \\ & S \succeq 0, \end{aligned}$$

onde $x \in \mathbb{R}^n$ e $S \in S(s)$ são as variáveis.

Consideremos também o seu problema dual na forma (2.27):

$$\begin{aligned} & \max \ tr\,(A_0 Z) \\ \text{s.a } & -tr\,(A_i Z) = c_i, \forall i = 1, \ldots, n \\ & Z \succeq 0. \end{aligned}$$

Teorema 3 *(Condições Suficientes de optimalidade) Sejam (x^*, S^*) solução admissível do problema primal de SDP (2.17) e Z^* solução admissível do problema dual (2.27). Se as seguintes condições estão satisfeitas,*

1. $\sum_{i=1}^{n} A_i x_i^* + S^* = -A_0$
2. $-tr\,(A_i Z^*) = c_i,\ i = 1, ..., n$
3. $S^* Z^* = 0$

então (x^*, S^*) é solução óptima de (2.17) e (Z^*), solução óptima de (2.27).

A condição 1 é chamada condição de admissibilidade primal; à condição 2 chamamos condição de admissibilidade dual. A condição 3 é usualmente designada condição de complementaridade (*"complementary slackness"*).

Em Klerk [27], está demonstrado que quando a condição de Slater está satisfeita, então as condições do Teorema 3 são condições necessárias e suficientes de optimalidade.

Teorema 4 *Supondo que a condição de Slater é satisfeita tanto no problema primal (2.17) como no dual (2.27), então o sistema de condições 1-3 do Teorema 3 reúne as condições necessárias e suficientes para que as soluções admissíveis (x^*, S^*) e Z^* dos problemas primal e dual, respectivamente, sejam soluções óptimas para estes problemas.*

De acordo com o que foi dito e com Bonnans e Shapiro [7], as condições de optimalidade para um problema primal de SDP podem ser formuladas da seguinte forma:

Teorema 5 *Supondo que é satisfeita a condição de Slater para as restrições do problema de SDP (2.15), então $x^* \in \mathbb{R}^n$ é solução óptima deste problema se e só se existe uma matriz $Z^* \in \mathcal{P}(s)$ tal que*

$$tr\left(Z^* A_j\right) + c_j = 0, j = 1, ..., n$$
$$tr\left(Z^* \mathcal{A}(x^*)\right) = 0,$$

onde s é a dimensão das matrizes quadradas A_j, com $j = 1, ..., n$.

Exemplo 13 *Consideremos o problema*

$$\min \ -2x$$

$$s.a \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \preceq 0$$

Vamos verificar que a solução admissível $x^* = 1$ é solução óptima do problema. Neste exemplo $s = 2$ e $n = 1$.

De facto, seja uma matriz $Z^* = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \in \mathcal{P}(2)$.

Neste caso, $\mathcal{A}(x^*) = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$.

Vamos verificar se as condições do Teorema 5 são satisfeitas.

$$tr\left(Z^* A_1\right) + c = 0$$
$$tr\left(Z^* \mathcal{A}(x^*)\right) = 0.$$

Substituindo, obtemos

$$tr\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}\right) = 0$$

e

$$tr\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right) - 2 = 0, \text{ como queriamos verificar.}$$

Portanto, $x^* = 1$ é solução óptima do problema.

Se o problema de SDP não satisfaz a condição de Slater, então as condições do Teorema 5 não garantem a optimalidade e são necessários estudos especiais.

Podemos concluir que a condição de Slater é uma condição importante para a caracterização de optimalidade.

2.5 Regularidade de problemas de Programação Semidefinida

Vimos anteriormente que não se consegue garantir a dualidade forte em SDP sem que se considerem satisfeitas certas condições de regularidade, como por exemplo, a condição de Slater (Freund e Sun [13]).

Segundo Klerk [27], para garantir a dualidade forte para problemas não regulares (em que não é satisfeita qualquer condição de regularidade), é possível aplicar um processo chamado regularização. O procedimento de regularização começa com uma solução admissível inicial e reduz o programa primal, num número finito de passos, a um programa regular. O procedimento faz uso da noção de cone minimal de $\mathcal{P}(s)$, $\mathcal{P}^f(s)$ (ver Gruber, Rendl e Wolkowicz [20] e Ramana, Tunçel e Wolkowicz [51] onde se podem encontrar todas as justificações para este processo de regularização).

Recentemente, tem-se dado especial atenção à classificação de problemas de SDP do ponto de vista da regularidade (ver Freund, Ordóñez e Toh [12], Gruber, Rendl e Wolkowicz [20], Jansson [24], Jansson et al. [25], Kostyukova e Tchemisova [32]).

Segundo Jansson [24], são frequentes situações em que os dados do problema podem não ser totalmente conhecidos e/ou precisos, e isto pode conduzir a termos problemas designados usualmente por *ill-posed* ("mal colocados"ou "mal comportados"). Um problema *well-posed* é usualmente tido como um problema que admite uma única solução óptima e que essa solução depende continuamente dos dados, caso contrário, o problema é *ill-posed*. Um problema *ill-conditioned* ("mal condicionado") é um problema em que um erro inicial dos dados conduz a resultados erróneos.

Há diferentes caracterizações ou definições para um problema ser *ill-posed*: em Freund, Ordóñez e Toh [12], os problemas que apresentam número de condição de Renegar infinito, $C(d) = \infty$, são considerados *ill-posed* e em Wolkowicz [66], o autor define que um problema é *ill-posed* se não verificar a condição de Slater. No entanto, no trabalho de Jansson et al. [25], os problemas são considerados *ill-posed* se o limite superior para o valor óptimo do problema, $\bar{p^*}$, for infinito. Neste artigo é dada especial atenção aos problemas que o autor considera *ill-posed* e que não satisfazem a condição de Slater.

Embora não se tenha conseguido encontrar resultados que estabelecem uma relação explícita entre condição de Slater e *ill-posedness* dos problemas de SDP, existem indicações de que esta ligação se verifica. O estudo entre duas definições de regularidade para problemas de SDP, do ponto de vista da satisfação da condição de Slater, e do ponto de vista da definição de *ill-posedness* dada por Jansson, é um dos objectivos deste trabalho.

2.6 Aplicações de Programação Semidefinida

Nas últimas décadas, tem-se verificado que a SDP se aplica a uma grande variedade de áreas de investigação, incluindo a teoria dos grafos, geometria, sistemas e teoria do controlo, optimização combinatória, computação quântica, data mining, algoritmos de aproximação,

estatística (ver Goemans [16], Helmberg [21], Klerk [27], Todd [54], Vandenberghe e Boyd [59]).

Nesta secção, fazemos referência a algumas das aplicações e reunimos alguns exemplos que ilustram a aplicabilidade de SDP nos dias que correm.

2.6.1 Problemas Combinatórios

A SDP é uma ferramenta importante para o desenvolvimento de algoritmos de aproximação para problemas combinatórios $\mathcal{NP}-$difíceis (para consulta detalhada sugerimos os compêndios Goldreich [18] e Viggo et al. [62]). O primeiro algoritmo de aproximação que usa SDP deve-se a Goemans e Williamson (1995) e foi construído para resolver o problema do corte máximo (max-cut) (ver Goemans [16]).

Dado um grafo G, pretende-se determinar uma partição do conjuntos dos seus vértices (através de um corte) em dois subconjuntos de forma a maximizar o número de arestas que ligam vértices destes subconjuntos.

Uma versão deste problema, chamada corte de peso máximo (weighted max-cut), considera que cada aresta tem um determinado peso associado e o objectivo é maximizar, não o número de arestas, mas o peso total das arestas com extremos em subconjuntos de vértices diferentes.

O exemplo que se segue foi retirado de Goemans [16].

Exemplo 14 *Consideremos um grafo $G = (V, E)$, onde V corresponde ao conjunto dos vértices e E é o conjunto das arestas do grafo. Seja ω_{ij} o peso associado à aresta $\{i, j\}$, onde $i, j \in V$.*

Pretendemos determinar uma partição do conjunto V de vértices em dois subconjuntos, S e $\bar{S}$, tal que $V = S \cup \bar{S}$ e $S \cap \bar{S} = \varnothing$.

Modelo de resolução:

O problema do corte de peso máximo pode ser formulado como um problema de Programação Quadrática inteira.

Se $y_i = 1$ para $i \in S$ e $y_i = -1$ caso contrário, então o valor do corte de peso máximo $\delta(S, \bar{S}) = \frac{1}{2} \sum_{\{i,j\} \in E} \omega_{ij} (1 - y_i y_j)$.

Assim, a formulação como problema de Programação Quadrática inteira é

$$\begin{aligned} \max\ & \delta(S, \bar{S}) = \frac{1}{2} \sum_{\{i,j\} \in E} \omega_{ij}(1 - y_i y_j) \\ s.a\ & y_i = 1 \text{ para } i \in S \\ & y_i = -1 \text{ para } i \in \bar{S} \end{aligned} \tag{2.31}$$

Podemos formular o problema do corte de peso máximo como um problema de SDP.

Consideremos a matriz $U = [u_{ij}]$, *com* $u_{ij} = y_i y_j$, *onde* $i, j \in V$ *e, tendo em conta que* $y_i = 1$ *para* $i \in S$ *e* $y_i = -1$ *caso contrário, todos os elementos da diagonal da matriz* U *são iguais a* 1. *Em Goemans [16] está justificado porque esta matriz* U *é semidefinida positiva.*

Assim, de acordo com Goemans [16], a formulação como problema de SDP é

$$\begin{aligned} \max\ \delta(S, \bar{S}) &= \frac{1}{2} \sum_{\{i,j\} \in E} \omega_{ij}(1 - u_{ij}) \\ s.a\ u_{ii} &= 1, \quad \forall i \in V \\ U &= [u_{ij}] \succeq 0, i, j \in V. \end{aligned} \tag{2.32}$$

2.6.2 Análise de Clusters e Data Mining

Um outro exemplo prático da aplicação de SDP, que surge em *data mining*, é em análise de crédito. Tomar a decisão se uma empresa é merecedora de crédito ou não, tem os seus riscos, e é evidente que é importante diminuir o risco na concessão de crédito.

O exemplo que se segue, adaptado de Konno et al. [31], ilustra uma forma possível de prever se uma empresa é solvente ou insolvente. Este exemplo está relacionado com o problema de separação óptima entre duas classes.

Exemplo 15 *Consideremos que temos uma base de dados financeiros relativos a empresas já catalogadas como solventes ou insolventes. Vamos usar estes dados para conseguir predizer, a partir de um novo exemplo, se a empresa é solvente ou insolvente. Assim, a partir dos dados de que dispomos, vamos resolver o problema de maneira a que os resultados obtidos possam ser usados como base para uma classificação de um grande número de empresas de forma automática.*

Modelo de resolução:

Se se representar todas as empresas na forma de pontos de $\mathbb{R}^n$, *a ideia de resolução consiste em encontrar uma superfície de separação que divida o conjunto dos pontos em dois grupos.*

Essa superfície separadora pode ser um hiperplano, caso os dados sejam linearmente separáveis. Mas, pode não existir um hiperplano que separe linearmente dois grupos. Assim, torna-se necessário encontrar uma fronteira de decisão não linear.

Consideremos A *a classe associada a empresas solventes e* B *a classe associada a empresas insolventes. Sejam* $a_i \in \mathbb{R}^n$, *com* $i = 1, ..., m$, *e* $b_l \in \mathbb{R}^n$, *com* $l = 1, ..., h$, *os respectivos vectores de dados financeiros.*

Se os dados forem linearmente separáveis, existe um vector $(c, c_0) \in \mathbb{R}^{n+1}$ *tal que*

$$c^T a_i > c_0, i = 1, ..., m \tag{2.33}$$
$$c^T b_l < c_0, l = 1, ..., h. \tag{2.34}$$

Então o seguinte hiperplano separa as duas classes:

$$H = \left\{ x \in \mathbb{R}^n : c^T x = c_0 \right\}.$$

As condições (2.33) e (2.34) são equivalentes a

$$c^T a_i \geq c_0 + 1, i = 1, ..., m$$
$$c^T b_l \leq c_0 - 1, l = 1, ..., h. \tag{2.35}$$

Consideremos uma margem em torno da superfície separadora que permita evidenciar a separação das classes, e que, enquanto treinamos o classificador, permita que haja alguns erros na classificação de empresas. Assim,

$c^T a_i < c_0 + 1, i = 1, ..., m$, *serão as empresas da classe A que são classificadas como pertencentes a B,*

$c^T b_l > c_0 - 1, l = 1, ..., h$ *serão as empresas da classe B que são classificadas como pertencentes a A.*

À má classificação dos dados vamos associar os seguintes valores:

dados $a_i, i = 1, ..., m$*:* y_i *é a distância de* a_i *ao hiperplano* $c^T x = c_0 + 1$,

dados $b_l, l = 1, ..., h$*:* z_l *é a distância de* b_l *ao hiperplano* $c^T x = c_0 - 1$.

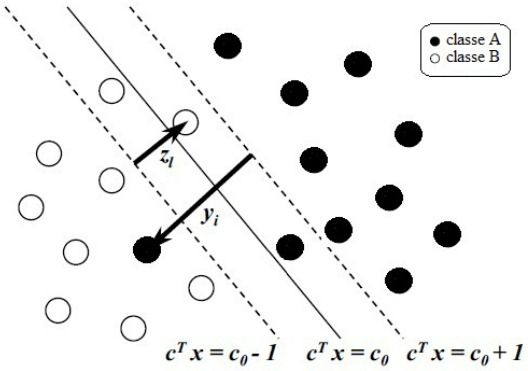

Figura 2.1: Hiperplano discriminante com uma margem para dados mal classificados.
(Adaptada de Konno et al. [31])

Vamos minimizar a soma dos erros de classificação e portanto, podemos formular o problema como um problema de LP na forma

$$\begin{aligned} &\min(1-\lambda)\frac{1}{m}\sum_{i=1}^{m} y_i + \lambda\frac{1}{h}\sum_{l=1}^{h} z_l \\ s.a\ &a_i^T c + y_i \geq c_0 + 1, \qquad i = 1, ..., m \\ &b_l^T c + z_l \leq c_0 - 1, \qquad l = 1, ..., h \\ &y_i \geq 0, \qquad i = 1, ..., m \\ &z_l \geq 0, \qquad l = 1, ..., h, \end{aligned} \tag{2.36}$$

onde $\lambda \in]0, 1[$ *, é uma constante que representa a importância relativa ao custo associado a uma má classificação de empresas.*

As variáveis neste problema são $c, c_0, y_1, ..., y_m, z_1, ..., z_h$.

Supomos agora que os dados não são linearmente separáveis. Neste caso pretendemos encontrar uma superfície de separação que seja quadrática.

Neste caso, o problema de encontrar a superfície separadora é dado por

$$\begin{aligned} &\min(1-\lambda)\frac{1}{m}\sum_{i=1}^{m} y_i + \lambda\frac{1}{h}\sum_{l=1}^{h} z_l \\ s.a\ &a_i^T D a_i + a_i^T c + y_i \geq c_0 + 1, \qquad i = 1, ..., m \\ &b_l^T D b_l + b_l^T c + z_l \leq c_0 - 1, \qquad l = 1, ..., h \\ &y_i \geq 0 \\ &z_l \geq 0, \end{aligned} \tag{2.37}$$

onde $D \in \mathcal{S}(n)$.

Aqui, a superfície separadora é dada por

$$Q = \left\{ x \in \mathbb{R}^n : x^T D x + x^T c = c_0 \right\},$$

e y_i *e* z_l *são as variáveis que representam as distâncias dos exemplos mal classificados às superfícies quadráticas* $x^T D x + x^T c = c_0 + 1$ *e* $x^T D x + x^T c = c_0 - 1$*, respectivamente.*

Para evitar que a configuração da superfície de separação seja muito complicada e que se gere uma região que conduza a um sobreajustamento dos dados, vamos exigir que uma das regiões criadas pela superfície de separação seja convexa.

Para isso vamos exigir que D *seja semidefinida positiva ou negativa. Se* $D \succeq 0$*, então vai ser gerado um elipsóide (ver [6]) que contém os exemplos de empresas insolventes no seu interior; se* $D \preceq 0$*, então vai ser gerado um elipsóide que contém os exemplos de empresas solventes no seu interior.*

Podemos então formular o problema de classificação como um problema de SDP na forma

$$\begin{aligned} & \min(1-\lambda)\frac{1}{m}\sum_{i=1}^{m} y_i + \lambda\frac{1}{h}\sum_{l=1}^{h} z_l \\ s.a\ & a_i^T D a_i + a_i^T c + y_i \geq c_0 + 1, \quad i = 1, ..., m \\ & b_l^T D b_l + b_l^T c + z_l \leq c_0 - 1, \quad l = 1, ..., h \\ & y_i \geq 0 \\ & z_l \geq 0 \\ & D \preceq 0 \end{aligned} \tag{2.38}$$

A última restrição permite gerar um elipsóide que contém $a_i, i = 1, ..., m$, no interior.

Desta forma, conseguimos gerar um elipsóide que engloba os exemplos classificados como solventes, ou seja, exemplos que pertençam à classe A. Os exemplos que não se encontram no interior do elipsóide, serão classificados como empresas insolventes (classe B).

2.6.3 Engenharia e Controlo

Exemplo 16 *Muitos problemas de engenharia e controlo envolvem a determinação de valores próprios de matrizes simétricas.*

Por exemplo, numa aplicação de controlo, o maior valor próprio da matriz representativa do sistema dinâmico representa a estabilidade do sistema, e portanto, é desejável minimizar esse valor de modo a garantir uma maior estabilidade [1]*, enquanto que numa aplicação de engenharia e análise estrutural, o menor valor próprio da matriz do modelo pode representar a carga de compressão que o material suporta sem que haja ruptura, e então, será desejável maximizar esse valor.*

Assim, o problema de minimizar o maior valor próprio de uma função matricial $\mathcal{A}(x)$, definida em (2.16), pode ser formulado como um problema convexo (ver Boyd et al. [9]) de SDP:

$$\begin{aligned} & \min \lambda \\ s.a\ & \lambda I - \mathcal{A}(x) \succeq 0, \end{aligned} \tag{2.39}$$

onde $x \in \mathbb{R}^n$ e $\lambda \in \mathbb{R}$ são variáveis.

[1] Um sistema é estável se os valores próprios da matriz representativa do sistema pertencem ao semiplano complexo negativo.

De forma análoga, o problema de maximizar o menor valor próprio pode ser formulado como

$$\begin{aligned} &\max \lambda \\ &s.a\ \mathcal{A}(x) - \lambda I \succeq 0. \end{aligned} \tag{2.40}$$

2.7 Métodos de resolução de problemas de SDP

Há vários métodos que podem ser usados na resolução de programas semidefinidos. Neste trabalho reunimos informação sobre um dos métodos que têm sido citados por diversos autores, como Helmberg [21], Helmberg et al. [22], Wolkowicz et al. [67], Wolkowicz [65] e no site [69]. Nas mesmas referências podem ser encontradas descrições de outros métodos, como por exemplo, o método dos elipsóides, o método Bundle espectral e o método do lagrangeano aumentado generalizado.

Os problemas de SDP são resolvidos eficientemente por métodos de ponto interior. Tais métodos foram desenvolvidos primeiramente para LP, no início dos anos 80, por Karmarkar (ver Helmberg [21]). Após alguma controvérsia sobre o desempenho do método, diversos trabalhos mostraram que variações deste método apresentavam desempenho computacional superior ao Simplex. Mais tarde, já nos anos 90, Nesterov e Nemirovski desenvolveram uma teoria para problemas convexos gerais e Alizadeh, apresentou uma forma simples e unificada de estender os métodos de ponto interior de LP para SDP (ver [35]).

Nesta secção, vamos descrever um algoritmo que pode ser usado para resolver problemas de SDP: o método (de ponto interior) primal-dual apresentado em Helmberg et al. [22] (denominado abordagem segundo HRVW), que resolve os problemas primal e dual simultaneamente. Este método é também conhecido na literatura como o método XZ.

Resolvemos incidir sobre o método primal-dual segundo a direcção HRVW, porque experiências numéricas em [70] sugerem que este tipo de métodos apresenta bons resultados mesmo quando não está satisfeita a condição de Slater.

Apresentamos em A.1 um quadro que reflecte o resultado do estudo que fizemos, reunindo as experiências numéricas com o programa *CSDP* disponíveis na internet com os resultados que obtivemos sobre a regularidade dos problemas. A partir deste quadro, podemos verificar que as soluções determinadas pelo programa são muito próximas dos valores reais, independentemente do problema de SDP ser ou não regular (no sentido de satisfazer ou não a condição de Slater).

Método primal-dual de ponto interior

O método primal-dual de ponto interior pode ser desenvolvido através da aplicação do método de Newton às condições de optimalidade excluindo as restrições de não-negatividade

e incluindo uma perturbação, μ, nas condições de complementaridade.

Consideremos o problema primal de SDP na forma (2.17).

Os algoritmos de ponto interior começam com uma solução inicial escolhida dentro do cone das matrizes simétricas semidefinidas positivas, $\mathcal{P}(n)$. Para evitar sair do cone durante o processo de optimização, a resolução do programa original é substituída pela sequência de resoluções aproximadas de problemas auxiliares de barreira. Estes problemas admitem um termo de barreira na função objectivo, dado por $\mu \log det(S)$. A $\mu > 0$ chamamos parâmetro de barreira e a $\log det(S)$ chamamos função de barreira.

Seja o problema (2.17) modificado (ou perturbado) da seguinte forma:

$$\begin{aligned} & \min c^T x - \mu \log det(S) \\ \text{s.a } & \sum_{i=1}^{n} A_i x_i + S = -A_0 \\ & S \succeq 0. \end{aligned} \tag{2.41}$$

A função de Lagrange associada a este problema é

$$L(x, S, Z) = c^T x - \mu \log det(S) + tr\left(Z\left(\sum_{i=1}^{n} A_i x_i + A_0 + S\right)\right) \tag{2.42}$$

As condições de optimalidade de KKT, além de necessárias são também suficientes, uma vez que a função de Lagrange (2.42) é estritamente convexa (Pedregal [46]) e têm a forma

$$\begin{cases} \sum_{i=1}^{n} A_i x_i + A_0 + S = 0, \\ tr\,(A_i Z) + c_i = 0, i = 1, ..., n, \\ -\mu S^{-1} + Z = 0. \end{cases} \tag{2.43}$$

A condição $-\mu S^{-1} + Z = 0$ é equivalente a $ZS - \mu I = 0$, onde I é a matriz identidade de dimensão s. Esta condição chama-se condição de *complementaridade perturbada* (ver Wolkowicz [65]).

Considerando o sistema

$$\begin{cases} \sum_{i=1}^{n} A_i x_i + S = -A_0 \\ tr\,(A_i Z) = -c_i, i = 1, ..., n \\ S \succ 0, Z \succ 0 \\ ZS = \mu I \end{cases} \tag{2.44}$$

o conjunto $\{(x, S, Z) : 0 \leq \mu < \infty\}$ designa-se por trajectória central.

A trajectória central é uma curva suave e contínua. Para um $\mu > 0$ fixo, tem-se um único ponto $(x(\mu), S(\mu), Z(\mu))$ na trajectória central.

Suponhamos que (x^*, S^*) e Z^* são a solução óptima do primal e dual, respectivamente, e $(x(\mu), S(\mu), Z(\mu))$ um ponto que satisfaz (2.44). Em Helmberg [21] está demonstrado que

$$\lim_{\mu \to 0}(x(\mu), S(\mu), Z(\mu)) = (x^*, S^*, Z^*).$$

Dado um ponto da trajectória central, podemos obter o valor de μ.

Aplicando a operação traço a ambos os membros da condição de complementaridade perturbada, obtemos

$$tr(ZS) = tr(\mu I) = s\mu,$$

onde s é a dimensão de S e de Z.

Assim,

$$\mu = \frac{tr(ZS)}{s}. \tag{2.45}$$

O método começa pela escolha de uma tolerância ε e de um ponto (x^0, S^0, Z^0), com $S^0 \succ 0$ e $Z^0 \succ 0$, na vizinhança da trajectória central.

Na $k-$ésima iteração tem-se $x^k = x^k(\mu)$, $S^k = S^k(\mu)$, $Z^k = Z^k(\mu)$ a aproximação corrente.

O valor corrente de μ^k é calculado usando (2.45) com uma pequena diferença: dividimos a expressão por 2. Segundo Helmberg et al. [22], a experiência de LP sugere que esta pequena alteração permite um bom desempenho. Então,

$$\mu^k = \frac{tr(Z^k S^k)}{2s}.$$

e verifica-se o critério de paragem: $tr(Z^k S^k) \leq \varepsilon$.

Se o critério de paragem é satisfeito, o algoritmo termina e a solução corrente é a solução do problema.

Se o critério não está satisfeito, encontra-se uma direcção admissível $\triangle d = (\triangle x, \triangle S, \triangle Z)$ a partir da iterada corrente, de tal forma que o novo ponto $(x^k + \triangle x, S^k + \triangle S, Z^k + \triangle Z)$ caia na trajectória central. Para isso resolve-se pelo método de Newton o sistema de equações não lineares (2.44) que pode ser reformulado da seguinte forma:

$$\begin{gathered} R_P := \sum_{i=1}^{n} A_i x_i^k + A_0 + S^k = 0 \\ R_D := tr\left(A_i Z^k\right) + c_i = 0, i = 1, ..., n \\ R_C := Z^k S^k - \mu I = 0 \end{gathered}$$

Linearizando este sistema, obtemos

$$\begin{cases} \sum_{i=1}^{n} A_i \triangle x_i + \triangle S = -R_P \\ tr\left(A_i \triangle Z\right) = -R_D, i = 1, ..., n \\ Z^k \triangle S + \triangle Z S^k = -R_C. \end{cases} \tag{2.46}$$

Como em geral, as matrizes S e Z não comutam (ver Helmberg [21]), representa-se $\triangle S$ na forma $\triangle S = \frac{\triangle\hat{S}+\triangle\hat{S}^T}{2}$. Trata-se da abordagem "HRVW". Esta abordagem permite que $\triangle S$ não seja necessariamente simétrica. A ideia é substituir esta expressão em (2.46) e resolver o sistema resultante, obtendo a direcção $\triangle d = (\triangle x, \triangle S, \triangle Z)$.

Para garantir que as matrizes S^k e Z^k sejam definidas positivas, determina-se um passo, denotando por α_P e α_D, (encontrados a partir de uma pesquisa linear) e então temos o novo ponto

$$(x^{k+1}, S^{k+1}, Z^{k+1}) = (x^k + \alpha_P \triangle x, S^k + \alpha_P \triangle S, Z^k + \alpha_D \triangle Z).$$

Actualiza-se o valor de μ e repete-se o processo até que o critério de paragem seja satisfeito. De acordo com Helmberg et al. [22], a convergência do método é polinomial.

2.8 Resolução numérica dos problemas de SDP

Existem diferentes *solvers* (comerciais ou não) para resolução de problemas de SDP.

A maior parte dos *solvers* utilizam o método primal-dual segundo a direcção HRVW descrito em 2.7, ou versões dele, como é o caso do SDPA, CSDP, SDPT3 e SeDuMi. Os programas baseados neste método apresentam resultados bastante satisfatórios.

Os *solvers* mais usados para resolução de problemas de SDP computacionalmente estão descritos na tabela A.2, que se encontra em anexo.

Capítulo 3

Subespaço de Índices Imóveis no estudo do problema linear de SDP

O objectivo principal desta dissertação é estudar a regularidade dos problemas de SDP. Além das conhecidas condições que regularidade, como por exemplo, a condição de Slater, existem diferentes definições de regularidade (ver Jansson [24], Klerk [27], Wolkowicz [66]).

Em Kostyukova e Tchemisova [32] foi introduzida a noção de subespaço de índices imóveis para problemas lineares de SDP e foi mostrado que este subespaço é nulo para problemas regulares (em que se verifica que a condição de Slater está satisfeita). Foi também apresentado um algoritmo de determinação deste subespaço.

Neste capítulo, vamos descrever o método de construção do subespaço de índices imóveis.

3.1 Definições e Resultados

Consideremos o problema (2.15) de SDP definido no capítulo anterior:

$$\begin{aligned} &\min c^T x \\ &\text{s.a } \mathcal{A}(x) \preceq 0, \end{aligned}$$

onde $x \in \mathbb{R}^n$ e $\mathcal{A}(x) := \sum_{j=1}^{n} A_j x_j + A_0$, e as matrizes $A_j \in \mathcal{S}(s)$, $s > 1$, para todo o $j = 0, 1, ..., n$.

O conjunto de soluções admissíveis para o problema (2.15) é

$$\mathcal{X} = \{x \in \mathbb{R}^n : \mathcal{A}(x) \preceq 0\}.$$

A restrição $\mathcal{A}(x) \preceq 0$ no problema de SDP (2.15) é equivalente ao sistema infinito de desigualdades

$$l^T \mathcal{A}(x) l \leq 0, \forall l \in \mathbb{R}^s. \tag{3.1}$$

Sem perda de generalidade, podemos supôr que para cada vector l em (3.1) está satisfeito $\|l\| = 1$, onde $\|.\|$ é uma qualquer norma vectorial. Aqui vamos usar a norma euclideana: $\|l\| = \sqrt{l_1^2 + ... + l_s^2},\ l \in \mathbb{R}^s$.

Vamos escrever o problema de SDP (2.15) na forma equivalente

$$\begin{aligned} &\min c^T x \\ &\text{s.a } l^T \mathcal{A}(x) l \leq 0, \forall l \in L, \end{aligned} \tag{3.2}$$

onde

$$L := \{l \in \mathbb{R}^s : \|l\| = 1\}. \tag{3.3}$$

O problema (3.2) é um problema de Programação Semi-Infinita (SIP), com um número infinito de restrições na forma

$$\varphi(x, l) = l^T \mathcal{A}(x) l \leq 0, l \in L,$$

com L definido em (3.3). Estas restrições são lineares em termos da variável x e quadráticas em termos do índice l. É evidente que (3.2) é um problema convexo.

O conjunto admissível do problema (3.2) tem a forma

$$\left\{x \in \mathbb{R}^n : l^T \mathcal{A}(x) l \leq 0, \forall l \in L\right\},$$

e coincide com o conjunto admissível do problema (2.15):

$$\mathcal{X} = \{x \in \mathbb{R}^n : \mathcal{A}(x) \preceq 0\} = \left\{x \in \mathbb{R}^n : l^T \mathcal{A}(x) l \leq 0, \forall l \in L\right\}.$$

Exemplo 17 *Dado um problema linear de SDP*

$$\begin{aligned} &\min_{x \in \mathbb{R}^2} x_1 \\ &s.a\ \mathcal{A}(x) \preceq 0, \end{aligned}$$

onde $\mathcal{A}(x) = A_1 x_1 + A_2 x_2 + A_0$, *com*

$$A_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},\ A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\ e\ A_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$

a sua formulação equivalente como um problema de SIP é

$$\begin{aligned} &\min_{x \in \mathbb{R}^2} x_1 \\ &s.a\ l^T \begin{bmatrix} x_1 & 1 \\ 1 & x_2 \end{bmatrix} l \leq 0, \forall l \in L, \end{aligned}$$

com L definido em (3.3), ou equivalentemente,

$$\begin{aligned} &\min_{x \in \mathbb{R}^2} x_1 \\ &s.a\ l_1^2 x_1 + l_2^2 x_2 + 2 l_1 l_2 \leq 0, \forall l \in L = \left\{l = (l_1, l_2) \in \mathbb{R}^2 : \|l\| = 1\right\}. \end{aligned} \tag{3.4}$$

Seguem-se algumas noções que são cruciais ao desenvolvimento do trabalho e que vamos introduzir de acordo com Kostyukova e Tchemisova [32].

Definição 19 *Um índice $l \in L$ é chamado imóvel relativamente às restrições do problema de SIP (3.2) se $l^T \mathcal{A}(x) l = 0, \forall x \in \mathcal{X}$.*

Vamos designar o conjunto de índices imóveis no problema (3.2) por L^* :

$$L^* = \left\{ l \in L : l^T \mathcal{A}(x) l = 0, \forall x \in \mathcal{X} \right\}. \tag{3.5}$$

Definição 20 *Dizemos que as restrições do problema semi-infinito (3.2) satisfazem a condição de Slater se*

$$\exists \bar{x} \in \mathbb{R}^n \quad tal \quad que \quad l^T \mathcal{A}(\bar{x}) l < 0, \forall l \in L. \tag{3.6}$$

Em [32], foi provado que se as restrições de um problema convexo de SIP satisfazem a condição de Slater, então o conjunto de índices imóveis é vazio, e o recíproco também é verdade. Formulamos a proposição (adaptada da *Proposição 2* em [32]) que nos permite afirmar o que foi dito.

Proposição 8 *Dado o problema convexo de SIP (3.2) tal que $\mathcal{X} \neq \varnothing$, então as seguintes condições são equivalentes:*

R1: As restrições do problema (3.2) satisfazem a condição de Slater.

R2: O conjunto de índices imóveis (3.5) do problema (3.2) é vazio: $L^* = \varnothing$.

Prova: Vamos provar que $R1 \Rightarrow R2$. Suponhamos que $\mathcal{X} \neq \varnothing$ e que as restrições do problema (3.2) satisfazem a condição de Slater. Então,

$$\exists \bar{x} \in \mathcal{X} \quad \text{tal} \quad \text{que} \quad l^T \mathcal{A}(\bar{x}) l < 0, \forall l \in L,$$

o que implica que, qualquer que seja $l \in L$, não se verifica que para todo $x \in \mathcal{X}$

$$l^T \mathcal{A}(x) l = 0.$$

Assim, $L^* = \varnothing$.

A implicação recíproca é demonstrada em [32]. ■

Como os problemas (2.15) e (3.2) são equivalentes, podemos transpôr a noção de índices imóveis para o problema de SDP. Formulamos então a proposição (*Proposição 3* em [32]):

Proposição 9 *Dado o problema de SDP na forma (2.15), as restrições deste problema satisfazem a condição de Slater se e só se o conjunto de índices imóveis L^* é vazio, onde L^* é definido por (3.5).*

Em [32], foi demonstrado que o conjunto L^* em (3.5) pode ser representado na forma:

$$L^* = L \cap \mathcal{M}, \tag{3.7}$$

onde $\mathcal{M}$ é um subespaço de espaço vectorial $\mathbb{R}^s$ dado por

$$\mathcal{M} := \left\{ l \in \mathbb{R}^s : l^T \mathcal{A}(x) l = 0, \forall x \in \mathcal{X} \right\} = \left\{ l \in \mathbb{R}^s : \mathcal{A}(x) l = 0, \forall x \in \mathcal{X} \right\}$$

e chamado *subespaço de índices imóveis* do problema de SDP (2.15).

Tendo em conta a Proposição 9, pode-se concluir que o problema de SDP (2.15) satisfaz a condição de Slater se e só se $\mathcal{M}$ é nulo.

Seja a dimensão de $\mathcal{M}$ igual a $s^* : dim(\mathcal{M}) = s^* \leq s$.

De acordo com [32], o conjunto admissível do problema (2.15) pode ser representado na forma

$$\mathcal{X} = \left\{ x \in \mathbb{R}^n : \mathcal{A}(x) m_i = 0, i = 1, ..., s^*, l^T \mathcal{A}(x) l \leq 0, \forall l \in \mathcal{M}^\perp \right\},$$

onde $\mathcal{M}^\perp$ é o complemento ortogonal do subespaço $\mathcal{M}$ em $\mathbb{R}^s$. A última representação do conjunto $\mathcal{X}$ pode ser reescrita na forma matricial

$$\mathcal{X} = \left\{ x \in \mathbb{R}^n : \mathcal{A}(x) M = 0, N^T \mathcal{A}(x) N \preceq 0 \right\},$$

onde $M = (m_i, i = 1, ..., s^*) \in \mathbb{R}^{s \times s^*}$ é a matriz básica do subespaço $\mathcal{M}$ e $N \in \mathbb{R}^{s \times p^*}$, com $p^* = s - s^*$, é a matriz básica de $\mathcal{M}^\perp$, complemento ortogonal de $\mathcal{M}$ em $\mathbb{R}^s$.

Consideremos o conjunto convexo $\mathcal{Q} = \left\{ x \in \mathbb{R}^n : \mathcal{A}(x) M = 0 \right\}$.

Então o problema de SDP (2.15) pode ser reformulado na forma equivalente

$$\begin{aligned} &\min_{x \in \mathcal{Q}} c^T x \\ &\text{s.a } N^T \mathcal{A}(x) N \preceq 0. \end{aligned} \tag{3.8}$$

A forma semi-infinita do problema anterior é

$$\begin{aligned} &\min_{x \in \mathcal{Q}} c^T x \\ &\text{s.a } l^T \mathcal{A}(x) l \leq 0, \forall l \in L \cap \mathcal{M}^\perp. \end{aligned} \tag{3.9}$$

De acordo com o *Lema 2* em [32], o problema de SDP (3.8) satisfaz a condição

$$\exists \bar{x} \in \mathcal{Q} : N^T \mathcal{A}(\bar{x}) N \prec 0.$$

Esta última condição é chamada em [32] condição do tipo de Slater e está provado que esta condição garante que as condições de optimalidade para os problemas equivalentes (2.15) e (3.8) são as seguintes:

Teorema 6 *(Teorema 3 em [32]) Uma solução admissível $x^* \in \mathbb{R}^n$ é óptima para o problema de SDP (2.15) se e só existem vectores $\theta^k \in \Theta(x^*)$ e $\gamma_i \in \mathbb{R}^s$, com $i = 1, ..., s^*$, tais que*

$$\sum_{k=1}^{p^*} \theta^{kT} N^T A_j N \theta^k + c_j + \sum_{i=1}^{s^*} \gamma_i^T A_j m_i = 0, j = 1, ..., n, \tag{3.10}$$

onde $\Theta(x^*) = \left\{\theta^k \in \mathbb{R}^{p^*} : \theta^k \neq 0, \theta^{kT} N^T \mathcal{A}(x^*) N \theta^k = 0\right\}$.

Note-se que o Teorema 6 é o critério de optimalidade para o problema (2.15) e este critério não exige qualificações de restrições. Por exemplo, não é necessário que o problema de SDP (2.15) satisfaça a condição de Slater.

Segue-se um exemplo para mostrarmos a aplicação do Teorema 6.

Exemplo 18 *Consideremos o seguinte problema linear de SDP :*

$$\begin{array}{c} \min\limits_{x \in \mathbb{R}} 2x \\ s.a\ \mathcal{A}(x) \preceq 0, \end{array} \tag{3.11}$$

onde $\mathcal{A}(x) = A_1 x + A_0$, *com* $A_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ *e* $A_1 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$.

Verifica-se que as restrições deste problema não satisfazem a condição de Slater, e que o subespaço de índices imóveis tem a forma

$$\mathcal{M} = \left\{l = (0, l_2) \in \mathbb{R}^2\right\}.$$

Consideremos $x^* = 0$, uma solução admissível. Vamos verificar para x^* as condições de optimalidade do Teorema 6.

Uma vez que a dimensão de $\mathcal{M}$ é igual a $s^* = 1$, podemos considerar que a matriz básica do subespaço $\mathcal{M}$ é

$$M = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

composta por um único vector, que vamos denotar por m_1.

Como $p^* = s - s^*$, neste caso $p^* = 1$ e podemos supôr que uma matriz básica do complemento ortogonal de $\mathcal{M}$ é

$$N = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Determinemos o conjunto $\Theta(x^*)$:

$$\begin{aligned} \Theta(x^*) &= \left\{\theta \in \mathbb{R} : \theta \neq 0, \theta^T N^T \mathcal{A}(x^*) N \theta = 0\right\} \\ &= \left\{\theta \in \mathbb{R} : \theta \neq 0, \theta^2 N^T \mathcal{A}(x^*) N = 0\right\}. \end{aligned}$$

Uma vez que

$$N^T\mathcal{A}(x^*)N = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1,$$

então,

$$\Theta\left(x^*\right) = \left\{\theta \in \mathbb{R} : \theta \neq 0, \theta^T\theta = 0\right\}$$

e portanto, somos levados a concluir que $\Theta\left(x^*\right) = \varnothing$.

De acordo com o Teorema 6, $x^* = 0$ *é solução óptima do problema (3.11) se e só se existe o vector* $\gamma = \left(\gamma_1, \gamma_2\right)^T \in \mathbb{R}^2$ *tal que*

$$c + \gamma^T A_1 m_1 = 0.$$

Substituindo, temos

$$2+\begin{bmatrix} \gamma_1 & \gamma_2 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0 \Leftrightarrow 2+\begin{bmatrix} -\gamma_2 & -\gamma_1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0 \Leftrightarrow 2-\gamma_1 = 0 \Leftrightarrow \gamma_1 = 2$$

e $\gamma_2 \in \mathbb{R}$.

Admitindo, por exemplo, que $\gamma_2 = 1$, *então, encontrámos um vector* $\gamma = (2,1)^T$ *que satisfaz (3.10), e portanto, de acordo com o Teorema 6, a solução admissível* $x^* = 0$ *é solução óptima para o problema de SDP (3.11).*

Note-se que a dimensão do subespaço $\mathcal{M}$ de índices imóveis reflecte o grau de irregularidade do problema de SDP:

- se $dim(\mathcal{M}) = s$, então o grau de irregularidade é máximo;
- se $dim(\mathcal{M}) = 1$, então o grau de irregularidade é mínimo.

Em [32] foi demonstrado que existem situações em que as condições clássicas de optimalidade podem não permitir concluir sobre a optimalidade de determinada solução admissível do problema de SDP na forma (2.15), mas as condições de optimalidade que usam os vectores básicos do subespaço de índices imóveis são eficientes.

Torna-se então necessário desenvolver um método que nos permita verificar se um dado problema de SDP é regular, ou seja, se satisfaz a condição de Slater, e no caso em que não é regular, encontrar a base do subespaço de índices imóveis.

Em [32] é apresentado um algoritmo que encontra uma base do subespaço de índices imóveis de um problema de SDP num número finito de passos. As autoras chamaram-lhe *Algoritmo DIIS - Algorithm of Determination of the Immobile Index Subspace.*

O estudo deste algoritmo e a sua implementação são os objectivos principais deste trabalho. O algoritmo foi implementado em MatLab e testado usando diversos problemas de SDP existentes na literatura.

3.2 Algoritmo DIIS

3.2.1 Descrição do Algoritmo DIIS

Dado um problema de SDP na forma (2.15), o Algoritmo DIIS determina uma base $M = (m_i, i = 1, ..., s^*)$ do subespaço de índices imóveis, $\mathcal{M} \subset \mathbb{R}^s$, $dim(\mathcal{M}) = s^*$.

Supomos que o problema de SDP linear na forma (2.15) tem o conjunto de soluções admissíveis, $\mathcal{X} = \{x \in \mathbb{R}^n : \ \mathcal{A}(x) \preceq 0\}$ não vazio [1] e $s > 1$, com $s \in \mathbb{N}$, onde s é a dimensão do espaço das matrizes simétricas do problema que se está a considerar.

Faça-se $I^1 = \emptyset$ e $k = 1$.

Iteração Geral: No início da k-ésima iteração do algoritmo, é conhecido um conjunto de vectores linearmente independentes $m_i \in \mathcal{M},\ i \in I^k$, onde I^k é o conjuntos dos índices construído na iteração anterior.

Faça-se $p_k = s - \left|I^k\right|$ e encontre-se a solução

$$l_i \in \mathbb{R}^s,\ i = 1, \ldots, p_k;\ \gamma_i \in \mathbb{R}^s,\ i \in I^k \tag{3.12}$$

do sistema

$$\begin{cases} \sum\limits_{i=1}^{p_k} l_i^T A_j l_i + \sum_{i \in I^k} \gamma_i^T A_j m_i = 0, \quad j = 0, 1, \ldots, n, \\ \sum\limits_{i=1}^{p_k} \|l_i\|^2 = 1, \\ l_i^T m_j = 0\ , \quad j \in I^k,\ i = 1, \ldots,\ p_k. \end{cases} \tag{3.13}$$

Este sistema consiste num número finito de equações.

Se o sistema (3.13) não tem solução, então os vectores m_i, com $i \in I^k$, formam uma base do subespaço $\mathcal{M}$ ([32]).

Caso contrário, considere-se a solução (3.12) do sistema (3.13). Seja $\left\{m_i, i \in \triangle I^k\right\}$, $\triangle I^k = \left\{\left|I^k\right| + 1, \ldots, \left|I^k\right| + s_k\right\}$, $s_k \geq 1$, é o subconjunto maximal dos vectores linearmente independentes do conjunto $\{l_i,\ i = 1, \ldots, p_k\}$.

Uma vez que, por construção,

$$\sum_{i=1}^{p_k} \|l_i\|^2 = 1\ ,$$

[1] O estudo da admissibilidade do problema está fora dos objectivos deste trabalho. No entanto, notemos que existem estudos que permitem verificar se um dado problema é ou não admissível, como por exemplo, [25, 48].

concluímos que $\triangle I^k \neq \emptyset$.

Faz-se $I^{k+1} = I^k \cup \triangle I^k$, $k = k + 1$ e repete-se a iteração.

É evidente que o algoritmo pára após um máximo de s iterações.

Suponha-se que o Algoritmo DIIS parou após k^* iterações, com $k^* \leq s$. Considerem-se os vectores m_i , com $i \in I^{k^*}$ e seja $s^* = \left|I^{k^*}\right|$ o número de vectores m_i. Pela sua construção, sabe-se que estes vectores são linearmente independentes. Denote-se por $\bar{\mathcal{M}} \subset \mathbb{R}^s$ o subespaço gerado pelos vectores anteriores.

O teorema que se segue (*Teorema 6* em [32]) prova que o conjunto $\bar{\mathcal{M}}$ coincide com o subespaço de índices imóveis $\mathcal{M}$ do problema.

Teorema 7 *Dado o problema de SDP na forma (2.15), com o conjunto de soluções admissíveis, $\mathcal{X}$, não vazio, o conjunto $\bar{\mathcal{M}}$, construído pelo Algoritmo DIIS, é o subespaço de índices imóveis no problema (2.15).*

3.2.2 Exemplos de aplicação do algoritmo DIIS

Exemplo 19 *Considere-se o problema de SDP, adaptado de Kostyukova e Tchemisova [32]:*

$$\begin{array}{ll} & \min\limits_{x\in\mathbb{R}^3} \quad x_1 + 2x_2 + 3x_3 \\ s.a & \left[\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right] x_1 + \left[\begin{array}{cc} 2 & 1 \\ 1 & 0 \end{array}\right] x_2 + \left[\begin{array}{cc} 3 & 1 \\ 1 & 0 \end{array}\right] x_3 + \left[\begin{array}{cc} -1 & -1 \\ -1 & 0 \end{array}\right] \preceq 0 \end{array} \tag{3.14}$$

Neste exemplo, $s = 2$ e

$$A_0 = \left[\begin{array}{cc} -1 & -1 \\ -1 & 0 \end{array}\right], A_1 = \left[\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right], A_2 = \left[\begin{array}{cc} 2 & 1 \\ 1 & 0 \end{array}\right] \text{ e } A_3 = \left[\begin{array}{cc} 3 & 1 \\ 1 & 0 \end{array}\right].$$

Vamos encontrar uma base M do subespaço de índices imóveis, usando o Algoritmo DIIS.

Seja $k = 1$, $I^1 = \emptyset$.

Calculamos $p_1 = s - \left|I^1\right|$. Logo, $p_1 = 2 - 0 = 2$.

Resolvemos o sistema:

$$\begin{cases} \sum\limits_{i=1}^{2} l_i^T A_j l_i = 0 \ , \quad j = 0, 1, 2, 3 \\ \sum\limits_{i=1}^{2} \|l_i\|^2 = 1 \end{cases} \tag{3.15}$$

em termos de vectores $l_1 = (l_{11}, l_{12})^T$ e $l_2 = (l_{21}, l_{22})^T$. Ou seja,

$$\begin{cases} l_1^T A_0 l_1 + l_2^T A_0 l_2 = 0 \\ l_1^T A_1 l_1 + l_2^T A_1 l_2 = 0 \\ l_1^T A_2 l_1 + l_2^T A_2 l_2 = 0 \\ l_1^T A_3 l_1 + l_2^T A_3 l_2 = 0 \\ \|l_1\|^2 + \|l_2\|^2 = 1. \end{cases}$$

Este sistema pode ser reescrito na forma mais explícita em termos de componentes de vectores l_1 *e* l_2.

$$\begin{cases} \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} l_{21} \\ l_{22} \end{bmatrix} = 0 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{21} \\ l_{22} \end{bmatrix} = 0 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{21} \\ l_{22} \end{bmatrix} = 0 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{21} \\ l_{22} \end{bmatrix} = 0 \\ {l_{11}}^2 + {l_{12}}^2 + {l_{21}}^2 + {l_{22}}^2 = 1 \end{cases},$$

de onde obtemos

$$\begin{cases} -{l_{11}}^2 - 2l_{11}l_{12} - {l_{21}}^2 - 2l_{21}l_{22} = 0 \\ l_{11}l_{12} + l_{21}l_{22} = 0 \\ {l_{11}}^2 + l_{11}l_{12} + {l_{21}}^2 + l_{21}l_{22} = 0 \\ 3{l_{11}}^2 + 2l_{11}l_{12} + 3{l_{21}}^2 + 2l_{21}l_{22} = 0 \\ {l_{11}}^2 + {l_{12}}^2 + {l_{21}}^2 + {l_{22}}^2 = 1 \end{cases}$$

Resolvendo este último sistema, obtemos

$$\begin{cases} l_{11} = 0 \\ l_{21} = 0 \\ l_{22}^2 = 1 - l_{12}^2 \\ l_{12} \in \mathbb{R} \end{cases}.$$

Supondo, por exemplo, que $l_{12} = 0.1$, *obtemos* ${l_{22}}^2 = 1 - 0.1^2$, *e logo* $l_{22} = 0.995$. *Assim, o sistema admite a solução*

$$l_1 = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix}, \quad l_2 = \begin{bmatrix} 0 \\ 0.995 \end{bmatrix}.$$

É evidente que o subconjunto maximal de vectores linearmente independentes em $\{l_1,\ \ l_2\}$ *é constituído por um único vector, logo* $s_1 = 1$ *e* $\triangle I^1 = \{|I^1| + 1\} = \{0 + 1\} = \{1\}$ *. Podemos supôr que* $m_1 = l_1$ *e, sendo assim,* $\left\{m_i,\ i \in \triangle I^1\right\} = \{m_1\} = \left\{\begin{bmatrix} 0 \\ 0.1 \end{bmatrix}\right\}$.

$I^2 = I^1 \cup \triangle I^1$, *portanto,* $I^2 = \emptyset \cup \{1\} = \{1\}$.

Seja agora $k = 2$.

Calculamos $p_2 = s - |I^2|$ *e então,* $p_2 = 1$.

Resolvemos o sistema nas incógnitas l_1 *e* γ_1*:*

$$\begin{cases} l_1^T A_0 l_1 + \gamma_1^T A_0 m_1 = 0 \\ l_1^T A_1 l_1 + \gamma_1^T A_1 m_1 = 0 \\ l_1^T A_2 l_1 + \gamma_1^T A_2 m_1 = 0 \\ l_1^T A_3 l_1 + \gamma_1^T A_3 m_1 = 0 \\ \|l_1\|^2 + \|l_2\|^2 = 1 \\ l_1^T m_1 = 0 \end{cases}$$

Substituindo, $l_1 = (l_{11}, l_{12})^T$, $\gamma_1 = (\gamma_{11}, \gamma_{12})^T$, *as matrizes* A_0, A_1, A_2, A_3 *e o vector* m_1, *obtemos*

$$\begin{cases} \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} \gamma_{11} & \gamma_{12} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} = 0 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} \gamma_{11} & \gamma_{12} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} = 0 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} \gamma_{11} & \gamma_{12} \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} = 0 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} + \begin{bmatrix} \gamma_{11} & \gamma_{12} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} = 0 \\ {l_{11}}^2 + {l_{12}}^2 = 1 \\ \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} = 0 \end{cases},$$

ou seja,

$$\begin{cases} -1 - 0.1\gamma_{11} = 0 \\ 0.1\gamma_{11} = 0 \\ 2 + 0.1\gamma_{11} = 0 \\ 3 + 0.1\gamma_{11} = 0 \\ {l_{11}}^2 = 1 \\ l_{12} = 0 \end{cases}$$

Do último sistema obtemos

$$\begin{cases} 0.1\gamma_{11} = -1 \\ 0.1\gamma_{11} = 0 \\ 0.1\gamma_{11} = -2 \\ 0.1\gamma_{11} = -3 \\ {l_{11}}^2 = 1 \\ l_{12} = 0 \end{cases}$$

e concluímos que o sistema não tem solução, por isso, STOP. A matriz do subespaço de índices imóveis é

$$M = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix}.$$

Logo, podemos concluir que as restrições do problema (3.15) não satisfazem a condição de Slater, uma vez que o subespaço de índices imóveis, $\mathcal{M}$, é não nulo.

Exemplo 20 *Considere-se agora o problema de SDP Linear:*

$$\begin{array}{ll} & \min\limits_{x\in\mathbb{R}^3} \quad 48x_1 - 8x_2 + 20x_3 \\ s.a & \begin{bmatrix} 10 & 4 \\ 4 & 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 & 0 \\ 0 & -8 \end{bmatrix} x_2 + \begin{bmatrix} 0 & -8 \\ -8 & -2 \end{bmatrix} x_3 + \begin{bmatrix} 11 & 0 \\ 0 & -23 \end{bmatrix} \preceq 0. \end{array} \tag{3.16}$$

Vamos mostrar que, neste caso, o subespaço de índices imóveis é vazio, ou seja, as restrições deste problema satisfazem a condição de Slater.

A dimensão das matrizes A_i , $i = 0, 1, 2, 3$, é 2; portanto, $s = 2$.

Seja $k = 1$, $I^1 = \emptyset$ e $M^1 = [\ ]$.

Calculamos $p_1 = s - |I^1| = 2$.

Resolvemos o sistema:

$$\begin{cases} \sum\limits_{i=1}^{2} l_i^T A_j l_i = 0 \ , \quad j = 0, 1, 2, 3 \\ \sum\limits_{i=1}^{2} \|l_i\|^2 = 1, \end{cases} \tag{3.17}$$

com $l_1 = (l_{11}, l_{12})^T$ e $l_2 = (l_{21}, l_{22})^T$.

Este sistema tem a forma

$$\begin{cases} l_1^T A_0 l_1 + l_2^T A_0 l_2 = 0 \\ l_1^T A_1 l_1 + l_2^T A_1 l_2 = 0 \\ l_1^T A_2 l_1 + l_2^T A_2 l_2 = 0 \\ l_1^T A_3 l_1 + l_2^T A_3 l_2 = 0 \\ \|l_1\|^2 + \|l_2\|^2 = 1 \end{cases}$$

e pode ser transformado na forma

$$\begin{cases} 11{l_{11}}^2 - 23{l_{12}}^2 + 11{l_{21}}^2 - 23{l_{22}}^2 = 0 \\ 10{l_{11}}^2 + 8l_{11}l_{12} + 10{l_{21}}^2 + 8l_{21}l_{22} = 0 \\ -8{l_{12}}^2 - 8{l_{22}}^2 = 0 \\ -16l_{11}l_{12} - 2{l_{12}}^2 - 16l_{21}l_{22} - 2{l_{22}}^2 = 0 \\ {l_{11}}^2 + {l_{12}}^2 + {l_{21}}^2 + {l_{22}}^2 = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} 11{l_{11}}^2 - 23{l_{12}}^2 + 11{l_{21}}^2 - 23{l_{22}}^2 = 0 \\ 10{l_{11}}^2 + 8l_{11}l_{12} + 10{l_{21}}^2 + 8l_{21}l_{22} = 0 \\ {l_{12}}^2 + {l_{22}}^2 = 0 \\ -16l_{11}l_{12} - 2{l_{12}}^2 - 16l_{21}l_{22} - 2{l_{22}}^2 = 0 \\ {l_{11}}^2 + {l_{12}}^2 + {l_{21}}^2 + {l_{22}}^2 = 1 \end{cases}$$

de onde obtemos

$$\begin{cases} l_{11} = 0 \\ l_{21} = 0 \\ l_{12} = 0 \\ l_{22} = 0 \\ {l_{11}}^2 + {l_{12}}^2 + {l_{21}}^2 + {l_{22}}^2 = 1 \end{cases}$$

O sistema não tem solução, por isso, STOP: o subespaço de índices imóveis $\mathcal{M}$ é nulo. O Algoritmo DIIS pára logo na primeira iteração, $k^* = 1$, com $I^{k^*} = \emptyset$.

Concluímos que as restrições do problema exposto satisfazem a condição de Slater, uma vez que o subespaço de índices imóveis, $\mathcal{M}$, é nulo. Assim, pode-se concluir que se trata de um problema regular.

3.2.3 Implementação do Algoritmo DIIS

O Algoritmo DIIS foi implementado em MatLab, por ser um ambiente simples de usar e muito versátil por ter determinadas funções matemáticas já implementadas que seriam úteis para o desenvolvimento do programa.

O programa desenvolvido é designado por *DIISalgorithm* e é constituído por uma rotina principal e duas auxiliares.

A rotina principal, *DIISalgorithm*, começa por ler o ficheiro em formato SDPA esparso da informação das matrizes A_i, com $i = 0, 1, ..., n$; chama a rotina auxiliar *nlsys1* para construir o sistema de equações não lineares para $k = 1$.

A resolução do sistema não linear é feita na rotina principal, que depois de encontrar uma solução, identifica os vectores linearmente independentes e armazena-os numa matriz, a tal matriz básica do subespaço de índices imóveis.

Numa qualquer próxima iteração, a rotina *DIISalgorithm* chama a rotina auxiliar *nlsys2* para construir o sistema de equações não lineares, onde já constam os vectores γ. O sistema construído vai ser resolvido a partir da função *fsolve* do MatLab que se encontra na rotina principal, e mais uma vez, após obter solução, identificam-se os vectores linearmente independentes com os vectores já inseridos na matriz básica.

No fim do procedimento, o programa devolve a matriz básica do subespaço de índices imóveis ou informa que o subespaço é nulo..

Capítulo 4

Experiências Numéricas

4.1 Problemas de teste

Nesta dissertação testámos com sucesso 57 problemas de SDP: alguns exemplos da base SDPLIB, outros da bibliografia.

A SDPLIB é uma colecção de problemas teste de SDP Linear, retirados de várias aplicações e de uma vasta bibliografia. Esta base foi tornada pública em 1999 e é actualizada pelo seu autor, Brian Borchers, cuja pesquisa se centrou nos métodos de pontos interiores para LP e SDP e em aplicações dessas técnicas para problemas de optimização combinatória. A biblioteca SDPLIB é de acesso livre na internet, a partir do site:

http://euler.nmt.edu/~brian/sdplib/sdplib .

A base SDPLIB é composta por 92 problemas de SDP Linear, e todos estão armazenados no formato esparso SDPA (são ficheiro no formato *dat-s*).

Os problemas usados para teste pertencem à base de dados de problemas SDPLIB e outros problemas foram retirados de diversas fontes: *K-Tn1,..., K-Tn8*, foram construídos a partir de problemas considerados em [32]; *tutSDPLIB*, encontra-se no tutorial da SDPLIB [8]; *tutSDPAM*, foi retirado do tutorial do *solver* SDPA-M, [14]; *helmberg1* de [21]; *polik1, polik2, polik3* de [47]; *VandBoyd1* de [59]; *LuoSturmZhang* retirado de [39]; *Todd* de [54]; *Mitchell2004* de [41]; *GaertnerMatousek1*, *GaertnerMatousek2* retirados de [15]; *YumingZhang1995* de [68]; *KojimaSDP2006* de [29]; *SturmZhang* de [53]; *FreundSun* retirado de [13] e os restantes foram adaptados de [30, 45, 46].

Referimos apenas 57 problemas, porque nestes casos conseguimos obter os resultados que estão apresentados na tabela 4.1, enquanto que os restantes exemplos da base SDPLIB que não constam nessa tabela, não nos foi possível obter um resultado usando o computador que tínhamos à disposição para o estudo; o programa devolveu "out of memory", e portanto não foi capaz de testar estas instâncias.

4.2 Testes de problemas de SDP com o Algoritmo DIIS

Os testes numéricos foram realizados num computador com sistema operacional Windows 7, processador Intel Core 2 Quad, Q8200 @ 2.33 GHz, com 4 GB de RAM e usando a versão do MatLab 7.10.0.499 (R2010a).

Os resultados dos testes realizados estão apresentados na tabela 4.1.
As colunas da tabela das experiências numéricas a partir do programa *DIISalgorithm*, tabela 4.1, representam o seguinte:

- *Problema*: Nome usado para o problema na base SDPLIB, ou atribuído neste trabalho;
- n: Número de variáveis x;
- s: Dimensão das matrizes A_i de restrições;
- $dim(\mathcal{M})$: Dimensão da base do subespaço de índices imóveis $\mathcal{M}$, encontrada pelo algoritmo;
- *iter*: Número de iterações realizadas pelo Algoritmo DIIS;
- $tempo(h : min : s : ms)$: Tempo computacional;
- *M é nulo*: significa que a Condição de Slater é satisfeita pelas restrições do problema.

Todas as experiências foram realizadas 10 vezes, porque o algoritmo gera aleatoriamente a aproximação inicial para a solução do sistema não linear a resolver em cada iteração, e foram registados os resultados referentes aos tempos mínimo e máximo.

As instâncias assinaladas com $*$ foram testadas com uma tolerância diferente da tolerância usada por defeito pelo programa, com o intuito do estudo ser o mais rigoroso possível.

Problema	n	s	$dim(\mathcal{M})$	*iter*	$tempo_{min}$	$tempo_{max}$	$\mathcal{M}$ *é nulo*
K-Tn1	1	2	1	2	1s76ms	1s81ms	×
K-Tn2	2	2	1	2	387ms	585ms	×
K-Tn3	3	2	1	2	122ms	135ms	×
K-Tn4	4	2	1	2	93ms	98ms	×
K-Tn5	5	2	1	2	329ms	341ms	×
K-Tn6	6	2	1	2	337ms	339ms	×
K-Tn7	7	2	1	2	361ms	370ms	×
K-Tn8	8	2	1	2	359ms	374ms	×
helmberg1	2	3	1	2	183ms	190ms	×
tutSDPLIB	2	4	0	1	79ms	92ms	√
tutSDPAM	3	2	0	1	39ms	43ms	√
example3isa	2	2	2	2	41ms	47ms	×
VandBoyd1	2	2	1	2	1s546ms	1s995ms	×
LuoSturmZhang	2	3	1	2	698ms	732ms	×
Todd	2	2	1	2	479ms	510ms	×
Mitchell2004	2	3	0	1	111ms	134ms	√
GaertnerMatousek1	2	4	0	1	999ms	1s78ms	√
GaertnerMatousek2	3	6	0	1	154s447ms	169s10ms	√
YumingZhang1995	4	4	3	2	812ms	844ms	×
KojimaSDP2006	4	3	0	1	128ms	193ms	√
Kojima1SDP2006	2	4	0	1	782ms	1s150ms	√
netLPtoSDP1	2	3	0	1	301ms	324ms	√
SturmZhang	3	5	0	1	1s376ms	1s593ms	√
FreundSun	2	3	0	1	375ms	424ms	√
polik1	1	2	1	2	2s578ms	3s126ms	×
polik2	2	3	1	2	852ms	856ms	×
polik3	1	2	1	2	41s55ms	48s940ms	×
LPtoSDP1	2	2	0	1	1s989ms	2s180ms	√
LPtoSDP2	2	3	0	1	381ms	403ms	√
LPtoSDP3	2	3	0	1	214ms	230ms	√
LPtoSDP4	3	2	0	1	1s7ms	1s320ms	√
LPtoSDP5	3	3	0	1	719ms	833ms	√
LPtoSDP6	2	3	0	1	99ms	122ms	√

Tabela 4.1: Resultados obtidos com o Algoritmo DIIS (problemas da bibliografia).

Problema	n	s	$dim(\mathcal{M})$	*iter*	$tempo_{min}$	$tempo_{max}$	$\mathcal{M}$ *é nulo*
control1	21	15	0	1	31s233ms	43s15ms	$\surd$
control2	66	30	0	1	6min46s178ms	8min29s348ms	$\surd$
control3	136	45	0	1	44min53s5ms	46min43s285ms	$\surd$
control4	231	60	0	1	2h38min27s213ms	2h39min19s555ms	$\surd$
**hinf1*	13	14	0	1	10s134ms	11s75ms	$\surd$
**hinf2*	13	16	0	1	24h31min58s777ms	24h32min31s602ms	$\surd$
hinf3	13	16	16	3	19min10s122ms	19min12s513ms	$\times$
hinf4	13	16	16	3	21min34s234ms	22min1s970ms	$\times$
hinf5	13	16	16	3	26min2s75ms	27min57s550ms	$\times$
hinf6	13	16	16	3	9min2s198ms	9min8s587ms	$\times$
hinf7	13	16	16	4	11min59s349ms	12min58s606ms	$\times$
hinf8	13	16	16	3	20min6s195ms	20min36s404ms	$\times$
hinf9	13	16	16	4	1h54min59s4ms	1h57min30s649ms	$\times$
hinf10	21	18	18	4	1h26min39s788ms	1h33min12s335ms	$\times$
hinf11	31	22	22	3	51min1s164ms	56min17s566ms	$\times$
hinf12	43	24	24	3	3h4min1s45ms	3h5min53s847ms	$\times$
**qap5*	136	26	0	1	29h46min58s25ms	29h49min22s367ms	$\surd$
**qap6*	229	37	0	1	30h20min3s1ms	30h21min59s785ms	$\surd$
**qap7*	358	50	0	1	31h6min21s3ms	31h8min59s352ms	$\surd$
**qap8*	529	65	0	1	36h57min45s898ms	36h59min58s692ms	$\surd$
theta1	104	50	0	1	28min56s1ms	30min23s210ms	$\surd$
truss1	6	13	0	1	2s33ms	2s118ms	$\surd$
truss3	27	31	0	1	1min58s347ms	2min3s460ms	$\surd$
truss4	12	19	0	1	10s121ms	10s851ms	$\surd$

Tabela 4.1: (cont.) Resultados obtidos com o Algoritmo DIIS (problemas da base de dados de problemas SDPLIB).

Analisando a tabela de resultados 4.1, podemos verificar que dos 57 problemas testados, 27 deles não satisfazem a condição de Slater.

Podemos também concluir que a dimensão do subespaço de índices imóveis nos fornece informação não só sobre a regularidade, mas também, no caso de um problema não ser regular, o grau de irregularidade de um problema de SDP. Assim, podemos verificar que todos os problemas da base SDPLIB, bem como o problema *example3isa*, apresentam grau de irregularidade máximo (igual ao valor de s para cada um dos problemas), enquanto que os problemas *K-Tn1, ..., K-Tn8*, *helmberg1*, *VandBoyd1*, *LuoSturmZhang*, *Todd, polik1*, ..., *polik3* apresentam grau de irregularidade mínimo, ou seja, grau 1. Somente o problema *YummingZhang1995* apresenta grau de irregularidade 3, que é inferior a $s = 4$.

Tolerâncias

Os problemas da tabela 4.1 assinalados com $*$, foram testados com várias tolerâncias, e apenas foram registados os resultados obtidos utilizando as tolerâncias $TolX = 10^{-12}$ e $TolFun = 10^{-15}$ por serem resultados mais refinados, enquanto que os restantes exemplos foram testados com tolerâncias definidas por defeito no programa, $TolX = 10^{-6}$ e $TolFun = 10^{-9}$.

Repetindo as experiências para diferentes valores de tolerâncias apenas os tempos computacionais foram crescendo à medida que as tolerâncias eram diminuídas, produzindo os mesmos resultados.

Resolução dos sistemas de equações não lineares

O programa *DIISalgorithm* usa a função *fsolve* do MatLab para resolver o sistema de equações não lineares através de Métodos Quasi-Newtonianos. Para aplicar esta função é necessário introduzir uma aproximação inicial. Esta aproximação inicial da solução do sistema pode ser introduzida pelo utilizador ou gerada automaticamente pelo programa. Os resultados apresentados na tabela 4.1 foram obtidos usando uma aproximação inicial gerada aleatoriamente pelo programa. As experiências efectuadas mostram que a escolha desta primeira aproximação não tem qualquer relevância nos resultados obtidos.

4.3 Testes de regularidade de problemas de SDP

O algoritmo DIIS permite verificar se um dado problema de SDP é regular.

Em 2.5 foi mencionado que existem estudos que indicam que as noções de não regularidade (quando a condição de Slater não é satisfeita) e *ill-posedness* estão interligadas.

A tabela que se segue permite-nos comparar os resultados obtidos na tabela 4.1 para os problemas da base SDPLIB com os estudos apresentados em Jansson et al. [25] e Freund, Ordóñez e Toh [12].

Problema	n	s	$dim(\mathcal{M})$	*regular DIISalgorithm*	*regular* $\bar{p}^*$	*regular C(d)*
control1	21	15	0	√	√	√
control2	66	30	0	√	√	√
control3	136	45	0	√	√	√
control4	231	60	0	√	√	√
**hinf1*	13	14	0	√	×	×
**hinf2*	13	16	0	√	√	√
hinf3	13	16	16	×	×	×
hinf4	13	16	16	×	×	×
hinf5	13	16	16	×	×	×
hinf6	13	16	16	×	×	×
hinf7	13	16	16	×	×	×
hinf8	13	16	16	×	×	×
hinf9	13	16	16	×	×	√
hinf10	21	18	18	×	×	×
hinf11	31	22	22	×	×	×
hinf12	43	24	24	×	×	×
**qap5*	136	26	0	√	×	×
**qap6*	229	37	0	√	×	×
**qap7*	358	50	0	√	×	×
**qap8*	529	65	0	√	×	×
theta1	104	50	0	√	√	√
truss1	6	13	0	√	√	√
truss3	27	31	0	√	√	√
truss4	12	19	0	√	√	√

Tabela 4.2: Estudo de regularidade de problemas de SDP usando o Algoritmo DIIS.

Em relação à tabela 4.2, podemos conjecturar que o significado de *ill-posedness* pode não ser o mesmo da não satisfação da condição de Slater.

Os resultados dos testes apresentados na tabela 4.2 mostram três caracterizações de regularidade que não são coincidentes. Estas conclusões seguem da comparação dos resultados obtidos (para os problemas constantes da base SDPLIB) com os estudos apresentados em Jansson et al. [25], onde são testados problemas de SDP e considerados *ill-posed* se o limite superior para o valor óptimo do problema $\bar{p}^*$ for infinito, e em Freund, Ordóñez e Toh [12], onde se considera que um problema é *ill-posed* se o número de condição de Renegar, $C(d)$, for infinito.

Apresentamos de seguida uma tabela que nos permite comparar, de uma forma mais visível, os nossos resultados de regularidade com os obtidos no estudo de Jansson et al. [25], onde os autores usam uma outra caracterização de regularidade.

O programa que implementámos, *DIISalgorithm*, testa a regularidade de um problema de SDP do ponto de vista de satisfação ou não da condição de Slater.

A tabela 4.3 reflecte os resultados dos teste dos problemas apresentados na tabela 4.2 do ponto de vista de regularidade (condição de Slater) e de *ill-posedness*, do ponto de vista da definição dada por Jansson et al. [25].

		Condição de Slater	
		Regular	*Não regular*
ill-posedness-Jansson	*Não ill-posed*	9	0
	Ill-posed	5	10

Tabela 4.3: Comparação de duas definições de regularidade de problemas de SDP.

Analisando esta última tabela, podemos concluir que as definições de não regularidade e *ill-posedness* segundo Jansson são diferentes, embora para a maioria dos exemplos estes fenómenos (não regularidade e *ill-posedness*) acontecem simultaneamente: dos 24 problemas, 9 são regulares e não *ill-posed* simultaneamente, e 10 problemas são não regulares e *ill-posed*, mas em 5 casos, o problema satisfaz a condição de Slater (é regular) e é *ill-posed*.

Assim, podemos conjecturar que possivelmente a noção de regularidade dada por Jansson é mais forte e é mais difícil de verificar. Precisaríamos de fazer mais testes e estudos teóricos para obter conclusões mais rigorosas sobre este assunto.

Também seria interessante comparar a regularidade (tendo em consideração a condição de Slater) com a noção de *ill-conditioning*. Este estudo não foi possível, uma vez que não existem resultados de testes numéricos publicados.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas as conclusões sobre o trabalho desenvolvido e as perspectivas de trabalho futuro.

5.1 Conclusões

Neste trabalho foi estudado um algoritmo que permite testar se um dado problema de SDP satisfaz a condição de Slater.

Os métodos que permitem obter boas aproximações da solução de problemas de SDP pressupõem que as restrições do problema satisfaçam uma condição de regularidade (a condição de Slater é uma das mais fortes), podendo assim aplicar as condições clássicas de optimalidade. Surge a questão: como proceder quando a condição de Slater não é satisfeita?

Em Kostyukova e Tchemisova [32] são apresentadas condições de optimalidade eficientes que não exigem que se verifique a condição de Slater. Estas condições estão baseadas nas noções de índices e subespaço de índices imóveis, noções associadas a problemas de SIP. Foram estudadas as bases teóricas que permitem transformar um problema de SDP num problema de SIP, bem como transpôr as noções de índices e subespaço de índices imóveis para o contexto de SDP. Mostrou-se que o conhecimento de uma matriz básica representativa do subespaço de índices imóveis permite concluir sobre a regularidade de um problema de SDP, do ponto de vista de satisfação da condição de Slater. Em Kostyukova e Tchemisova [32] é proposto um algoritmo que permite obter tal matriz, num número finito de passos. As autoras chamaram-lhe Algoritmo DIIS.

O objectivo fulcral deste trabalho era o estudo, implementação e teste do Algoritmo DIIS para testar a regularidade de problemas de SDP, do ponto de vista de satisfação da condição de Slater. Para os testes foram usados problemas de SDP Linear da literatura e da base de dados SDPLIB.

Não se conhecem outros estudos onde se teste a condição de Slater (ou outra qualquer condição de regularidade) para problemas de SDP.

O Algoritmo DIIS foi implementado em MatLab e denominado *DIISalgorithm*. A im-

plementação deste algoritmo permitirá testar a regularidade de problemas de SDP. O programa devolve a matriz básica do subespaço de índices imóveis quando as restrições do problema teste não satisfazem a condição de Slater e a sua dimensão permite saber o grau de irregularidade do problema; doutra forma, informa que o subespaço é nulo.

Os resultados das experiências mostraram que o programa *DIISalgorithm* permite testar problemas de SDP linear com dimensão até 500, dados os recursos computacionais que dispunhamos. Dos 57 problemas que se conseguiram testar, 27 são problemas não regulares.

Concluímos que as definições de não regularidade e *ill-posedness* segundo Jansson não são coincidentes, embora para a maioria dos exemplos testados estes fenómenos (não regularidade e *ill-posedness*) acontecem simultaneamente: dos 24 problemas, 9 são regulares e não *ill-posed* simultaneamente, e 10 problemas são não regulares e *ill-posed*, mas em 5 casos, o problema satisfaz a condição de Slater (é regular) e é *ill-posed*.

Podemos conjecturar que possivelmente a noção de regularidade dada por Jansson é mais forte e é mais difícil de verificar.

Os resultados e conclusões apresentados neste trabalho confirmam a potencialidade do programa *DIISalgorithm* no teste da regularidade de problemas de SDP.

5.2 Trabalho Futuro

A versão actual do programa *DIISalgorithm* demonstrou alguma dificuldade em obter resultados para problemas de SDP de larga escala, na medida em que o computador usado para o estudo não foi capaz de ler os problemas (out of memory).

Uma melhoria seria ter acesso a um computador com sistema operativo de 64bits, que permitiria usar mais memória RAM para tentar testar a regularidade de todos os problemas de SDP disponíveis, bem como aumentar a eficiência e rapidez no cálculo através de computação paralela.

Poderia também revelar-se um aperfeiçoamento, o uso outras linguagens de programação, como C, para aumentar a eficiência do programa.

No futuro pretendemos:

- aperfeiçoar o programa para conseguir testar todos os problemas de SDP disponíveis nas bases de dados de problemas,
- estudar a ligação entre problemas de SIP e SDP,
- continuar o estudo teórico da regularidade de problemas de SDP,
- desenvolver um algoritmo de resolução de problemas de SDP baseado nos estudos que apresentámos.

Bibliografia

[1] Alizadeh, F., *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM J.Opt., 5, 1995.

[2] Bastos, F. and Teixeira, A., *An extension of a variant of a predictor-corrector primal-dual method for linear programming to SDP*, 2005.

[3] Bazaraa, M.S., Sherali, H.D. and Shetty, C.M., *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons Inc., 1993.

[4] Beck, J., *A Holistic Approach to Sustainability Analysis of Industrial Networks*, Thesis, 2008.

[5] Bellman, R. and Fan, K., *On systems of linear inequalities in Hermitian matrix variables*, In: Convexity, Proceedings of Symposia in Pure Mathematics, Vol. 7, American Mathematical Society, Providence, RI, 1963.

[6] Bertoni, B., *Multi-dimensional Ellipsoidal Fitting*, preprint, Department of Physics, Southern Methodist University, 2010.

[7] Bonnans, J. F. and Shapiro, A., *Perturbation Analysis of Optimization Problems*, Springer-Verlag, New-York, 2000.

[8] Borchers, B., *SDPLIB 1.2, A library of Semidefinite Programming Test Problems*, Optimization Methods and Software, 1999.

[9] Boyd, S., El Ghaoui, L., Feron, E. and Balakrishnan, V., *Linear Matrix Inequalities in System and Control Theory*, SIAM studies in applied mathematics, Vol. 15, Philadelphia, 1994.

[10] Datorro, J., *Convex Optimization Euclidean Distance Geometry*, Meßoo, 2005.

[11] Donath, W. E. and Hoffman, A. J., *Lower bounds for the partitioning of graphs*, IBM J. of Research and Development, 17, 1973.

[12] Freund, R. M., Ordóñez, F. and Toh, K. C., *Behavioral Measures and their Correlation with IPM Iteration Counts on Semi-Definite Programming Problems*, Math. Programming, 109(2), 2007.

[13] Freund, R. M., Sun, J., *Semidefinite Programming I: Introduction and minimization of polynomials*, System Optimization, 2002.

[14] Fujisawa, K., Futakata, Y., Kojima, M., Matsuyama, S., Nakamura, S., Nakata, K. and Yamashita, M., *SDPA-M (SemiDefinite Programming Algorithm in MATLAB) User's Manual - Version 6.2.0*, Series B: Operations Research Department of Mathematical and Computing Sciences, May 2005.

[15] Gärtner, B. and Matousek, J., *Interior Point Methods*, Approximation Algorithms and Semidefinite Programming, http://www.ti.inf.ethz.ch/ew/courses/ApproxSDP09/, 2009.

[16] Goemans, M. X., *Semidefinite Programming and Combinatorial Optimization*, Documenta Mathematica, Extra Volume ICM 1998, Vol III, 1998.

[17] Goemans, M. X. and Williamson, D. P., *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, Journal of the ACM, 42, 1995.

[18] Goldreich, O., *P, NP, and NP-Completeness: The Basics of Complexity Theory*, Cambridge, 2010.

[19] Grotschel, M., Lovász, L. and Schrijver, A., *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[20] Gruber, G., Kruk, Rendl, F. and Wolkowicz, H., *Presolving for Semidefinite program without Constraint Qualifications*, in G. Gruber et al. editor, Proceedings of HPOP97, Second Workshop on High Perfomance Optimization Techniques, Rotterdam Netherlands, 1997.

[21] Helmberg, C., *Semidefinite Programming for Combinatorial Optimization*, ZIB Report, Berlin, 2000.

[22] Helmberg, C., Rendl, F., Vanderbei, R. J. and Wolkowicz, H., *An Interior-Point Method for Semidefinite Programming*, SIAM J. Optim., Vol. 6, No. 2, May 1996.

[23] Hettich, R. and Kortanek, K.O., *Semi-infinite programming: theory, methods and applications*, SIAM Rev. 35, 1993.

[24] Jansson, C., *Termination and Verification for Ill-Posed Semidefinite Programming Problems*, http://www.optimization-online.org/DB_ HTML/2005/06/1150.html, 2005.

[25] Jansson, C., Chaykin, D. and Keil, C., *Rigorous Error Bounds for the Optimal Value in SDP*, http://www.optimization-online.org/DB_ HTML/2005/01/1047.html, 2005.

[26] Jeyakumar, V. and Nealon, M. J., *Complete Dual Characterizations of Optimality for Convex Semidefinite Programming*, in "Constructive, Experimental and Nonlinear Analysis", vol. 27 of CMS Conference Proceedings American Mathematical Society, Providence, USA, 2000.

[27] Klerk, E. de, *Aspects of Semidefinite Programming - Interior Point Algorithms and Selected Applications*, Applied Optimization, Volume 65, Kluwer Academic Publishers, 2004.

[28] Klerk, E. de, Roos, C. and Terlaky, T., *A short survey on semidefinite programming*, Ten Years LNMB, PhD Research and Graduate Courses of the Dutch Network of Operations Research (W. K. K. H. et al., ed.), vol. 122, Amsterdam, The Netherlands: Centrum for Mathematics and Informatics (CWI), 1997.

[29] Kojima, M., *Introduction to Semidefinite Programs (Semidefinite Programming and Its Application)*, Institute for Mathematical Sciences National University of Singapore, 2006.

[30] Kolman, B. and Beck, R. E., *Elementary Linear Programming with Applications* , Elsevier Science Technology Books, 1995.

[31] Konno, H., et al, *Failure Discrimination by Semidefinite Programming*, Financial Engineering, Supply Chain and E-commerce, edited by P.Pardalos and V. Tsitsiringos, Kluwer Academic Publisher, Netherland, 2002.

[32] Kostyukova, O. I. and Tchemisova, T.V., *Optimality Criterion without Constraint Qualification for Linear Semidefinite Problems*, para aparecer em Journal of Mathematical Sciences (2011).

[33] Kostyukova, O. I. and Tchemisova, T.V., *Sufficient optimality conditions for convex Semi Infinite Programming*, Optimization Methods and Software, Volume 25, Issue 2, April 2010.

[34] Kostyukova, O. I., Tchemisova, T. V., and Yermalinskaya, S. A., *On the algorithm of determination of immobile indices for convex SIP problems*, IJAMAS-International Journal on Mathematics and Statistics 13, N^0 J08, 2008.

[35] Krishnan, K., *Linear Programming Approaches to Semidefinite Programming problems*, PhD thesis, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, July 2002.

[36] Krishnan, K. and Mitchell, J., *Semi-Infinite Linear programming approaches to SDP problems*, Fields Institute Communications Series, Volume 37, Novel approaches to hard discrete optimization problems, edited by P. Pardalos and H. Wolkowicz, 2003.

[37] Lovász, L., *On the Shannon capacity of a graph*, IEEE Transactions on Information Theory, 25, 1979.

[38] Lovász, L. and Schrijver, A., *Cones of matrices and set-functions and 0-1 optimization*, SIAM J. Optim., 1, 1991.

[39] Luo, Z., Sturm, J. and Zhang, S., *Duality results for conic convex programming*, Econometric Institute Report No. 9719/A, 1997.

[40] Mangasarian, O. L., *Nonlinear Programming*, SIAM'S Classics in Applied Mathematics, 1994.

[41] Mitchell, J. and Krishnan, K., *A unifying framework for several cutting plane methods for semidefinite programming*, Technical Report, Dept. of Computational and Applied Mathematics, Rice University, December 2003.

[42] Mittelmann, H. D., *An independent benchmarking of SDP and SOCP solvers*, Mathematical Programming, series B-95, 2003

[43] Nesterov, Y. E. and Nemirovski, A. S., *Conic formulation of a convex programming problem and duality*, Optim. Methods Software, vol 1, 1992.

[44] Nesterov, Y. E. and Nemirovski, A. S., *Interior Point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia, USA, 1994.

[45] Nocedal, J. and Wright, S. J., *Numerical Optimization*, Springer, 1999.

[46] Pedregal, P., *Introduction to Optimization*, Springer, 2004.

[47] Polik, I., *Semidefinite programming Feasibility and duality*, 2009.

[48] Polik, I. and Terlaky, T., *New stopping criteria for detecting infeasibility in conic optimization*, *AdvOL*-report #2007/09, 2007.

[49] Ramana, M. V., *An Exact Duality Theory for Semidefinite Programming and its Complexity Implications*, DIMACS Technical report 95-02R, RUTCOR, Rutgers University, New Brunswick, NJ, 1995.

[50] Ramana, M. V., *Introduction to Semidefinite Programming I: Basic properties and variations on the Goemans-Williamson approximation algorithm for max-cut*, Seminar on Semidefinite Programming, Amsterdam, May 2010.

[51] Ramana, M. V., Tunçel, L. and Wolkowicz, H., *Strong Duality for Semidefinite Programming*, SIAM J. Optimization, vol. 7, N 3, 1997.

[52] Seber, G. A. F., *Multivariate Observations*, Wiley, 2004.

[53] Sturm, J. and Zhang, S., *ON SENSITIVITY OF CENTRAL SOLUTIONS IN SEMIDEFINITE PROGRAMMING*, Mathematical Programming, Volume 90, Number 2, Springer, 1998.

[54] Todd, M. J., *Semidefinite Optimization*, in Acta Numerica, 2001, Cambridge University Press, Cambridge, UK, 2001.

[55] Tunçel, L., *Some Applications of Semidefinite Optimization from an Operations Research Viewpoint*, Iranian Journal of Operations Research 1, 2009.

[56] Tunçel, L. and Wolkowicz, H., *Strong Duality and Minimal Representations for Cone Optimization*, 9^{th} SIAM Conference on Optimization, Boston, May, 2008.

[57] Vandenberghe, L. and Boyd, S., *Applications of Semidefinite Programming*, Applied Numerical Mathematics, 29, 1999.

[58] Vandenberghe, L. and Boyd, S., *Connection between Semi-Infinite and Semidefinite Programming*, in Reemtsen, R. and Ruckmann, J. J., Eds, chapter 8 of *Semi-Infinite Programming*, Kluwer Academic Publishers, 1998.

[59] Vandenberghe, L. and Boyd, S., *Semidefinite Programming*, SIAM Review , vol 38, 1996.

[60] Vaz, A. I. F., *Aplicações, métodos e ferramentas para Programação Semi-Infinita Não Linear*, PhD thesis, 2003.

[61] Vieira, M. V. C., *Programação Semidefinida Positiva na Resolução de um Problema de Optimização Combinatória*, MSc thesis, 2001.

[62] Viggo, K., *Complexity and Approximation - Combinatorial optimization problems and their approximability properties*, list of problems available in http://www.csc.kth.se/∼viggo/wwwcompendium/ , 1999.

[63] Webb, A. R., *Statistical Pattern Recognition*, Second Edition, John Wiley Sons, 2002.

[64] Wolkowicz, H., *Duality for Semidefinite Programming*, Encyclopedia of Optimization, Part 4, 2009.

[65] Wolkowicz, H., *Semidefinite Programming*, 2004.

[66] Wolkowicz, H., *Strong Duality*, edited by Floudas, C. A. and Pardalos, P. M., in Encyclopedia of Optimization, Vol.6, Kluwer Academic Publishers, 2001.

[67] Wolkowicz, H., Saigal, R. and Vandenberghe, L., *Handbook of semidefinite programming: theory, algorithms, and applications*, Kluwer Academic Publishers, 2000.

[68] Zhang, Y., *Semidefinite Programming*, Lecture 2, 1995.

[69] http://sdpa.indsys.chuo-u.ac.jp/sdpa/intro.html#methods ,
visitado a última vez em 2010-09-30.

[70] http://euler.nmt.edu/∼brian/sdplib/sdplib.html ,
visitado a última vez em 2010-10-20.

[71] http://home.online.no/∼pjacklam/matlab/software/util/fullindex.html ,
visitado a última vez em 2010-04-26.

Apêndices

Apêndice A

Solvers de SDP

A.1 Resolução de Problemas de SDP

A tabela A.1 apresenta os resultados do nosso estudo sobre o uso do *solver* CSDP.

Problema	*Regular*	*Valor óptimo*	*Valor óptimo (CSDP)*
$control1$	$\surd$	$1.7784627e+01$	$1.7784627e+01$
$control2$	$\surd$	$8.3000000e+00$	$8.2999999e+00$
$control3$	$\surd$	$1.3633266e+01$	$1.3633266e+01$
$control4$	$\surd$	$1.9794230e+01$	$1.9794231e+01$
$hinf1$	$\surd$	$2.0325997e+00$	$2.032599709156590162e+00$
$hinf2$	$\surd$	$1.0967056e+01$	$1.096705562104867759e+01$
$hinf3$	$\times$	$5.6940778e+01$	$5.694077801037055764e+01$
$hinf4$	$\times$	$2.7476383e+02$	$2.747638302294833466e+02$
$hinf5$	$\times$	$3.62e+02$	$3.622133170906411124e+02$
$hinf6$	$\times$	$4.4892775e+02$	$4.489277453469505872e+02$
$hinf7$	$\times$	$3.90812e+02$	$3.908123265658610990e+02$
$hinf8$	$\times$	$1.16146e+02$	$1.161459857398232742e+02$
$hinf9$	$\times$	$2.3624926e+02$	$2.362492582529170022e+02$
$hinf10$	$\times$	$1.087118e+02$	$1.087118014611462939e+02$
$hinf11$	$\times$	$6.5862e+01$	$6.586208933139603516e+01$
$hinf12$	$\times$	$2e-1$	$1.861737913545270897e-10$
$qap5$	$\surd$	$-4.3600000e+02$	$-4.3600000e+02$
$qap6$	$\surd$	$-3.8143840e+02$	$-3.814384010555183409e+02$
$qap7$	$\surd$	$-4.2481971e+02$	$-4.248197092656959626e+02$
$qap8$	$\surd$	$-7.5695525e+02$	$-7.569552490964156277e+02$
$theta1$	$\surd$	$2.3000000e+01$	$2.3000000e+01$
$truss1$	$\surd$	$-8.9999963e+00$	$-8.999996315286969306e+00$
$truss3$	$\surd$	$-9.1099962e+00$	$-9.109996209202071427e+00$
$truss4$	$\surd$	$-9.0099963e+00$	$-9.009996291004547686e+00$

Tabela A.1: Estudo da eficiência do *solver* CSDP.

Os problemas testados com o programa CSDP pertencem à base de dados de problemas de SDP, SDPLIB e encontram-se disponíveis em [70].

Este estudo visa obter uma ideia do comportamento do programa CSDP quando um problema a resolver é não regular. Sabe-se que a maioria dos métodos de resolução de problemas de SDP partem do pressuposto que o problema é regular.

Analisando a tabela, podemos concluir que o programa CSDP apresenta resultados bastante satisfatórios, mesmo quando a condição de Slater não está satisfeita (segundo os resultados obtidos com o programa que desenvolvemos, *DIISalgorithm*).

A.2 Solvers de SDP

Nesta secção apresentamos uma breve compilação de características de alguns dos mais populares programas que permitem resolver problemas de SDP linear apresentados na forma (2.15).

Os programas $SDPA$, $CSDP$, $SeDuMi$ e $SDPT3$ são bastante estáveis e produzem bons resultados (ver Mittelmann [42]).

Em relação ao $DSDP$ parece ser um programa bastante competitivo com os referidos anteriormente.

Segundo Mittelmann [42], os programas $SDPA$ e $CSDP$ são bastante robustos e satisfazem as expectativas que o utilizador pode depositar em programas que se baseiam em métodos primal-dual.

Todos os *solvers* apresentados na tabela A.2 suportam o formato esparso SDPA para os dados de entrada. Este formato será explicado na secção B.3.

Os programas $SDPA$ e $CSDP$ são os únicos especificamente apropriados para resolver problemas de SDP.

Têm sido apresentadas versões destes programas (por exemplo, com uso de cálculo paralelo) por forma a melhorar a eficiência e desempenho na resolução de problemas.

De seguida apresentamos a tabela A.2 que reúne as principais características de alguns *solvers* de problemas de SDP.

Solver	*Características Principais*
SDPA	Usa um algoritmo baseado no *Mehrotra-type predictor-corrector* primal-dual method segundo a abordagem "HRVW"; apropriado para resolver problemas de SDP de pequena a média dimensão; implementação em C^{++}, com interface para MatLab; simples de usar; explora a estrutura esparsa dos dados de entrada; existe versão para cálculo paralelo, SDPARA, que permite resolver problemas de grande dimensão.
CSDP	Não é apropriado para *second order cone constraints*; o código usa uma versão predictor-corrector do método primal-dual de ponto interior de HRVW; apropriado para resolver problemas de SDP de pequena a média dimensão; suporta matrizes com estrutura diagonal por blocos; escrito em C e pode ser usado no MatLab; existe uma versão para cálculo paralelo, OPENMP.
SeDuMi	Permite resolver problemas de optimização com restrições lineares, quadráticas e semidefinidas; adequado para problemas de grande dimensão; explora a estrutura esparsa das matrizes; implementado em MatLab e C; usa o método primal-dual de ponto interior usando o esquema preditor-corrector; torna-se lento e pode esgotar a memória se os dados não têm estrutura diagonal por blocos.
DSDP	Programa implementado em C, com interface para o MatLab; apropriado para problemas de SDP; usa o *dual scaling potencial reduction method*; usa apenas a solução dual para gerar a direcção, poupando memória; explora também a estrutura diagonal por blocos.
SDPT3	Escrito em MatLab com algumas subrotinas em C e em Fortran; apropriado para problemas de programação cónica; usa um método primal -dual de ponto interior com um passo preditor-corrector e com as direcções HRVW ou de Newton; explora a estrutura esparsa e diagonal por blocos das matrizes; adequado para problemas de pequenas e médias dimensões, no entanto produziu bons resultados para problemas de grande dimensão.

Tabela A.2: Principais características dos *solvers* para problemas de SDP.

Apêndice B

Algoritmo DIIS no MatLab

B.1 Pseudo-código do Algoritmo DIIS em MatLab

Dado um problema de SDP na forma (2.15), o Algoritmo DIIS determina uma base $M = (m_i, i = 1, ..., s^*)$ do subespaço de índices imóveis, $\mathcal{M} \subset \mathbb{R}^n$, $dim(\mathcal{M}) = s^*$.

Suponha-se $s > 1$, com $s \in \mathbb{N}$.

Algoritmo DIIS

$k = 1$, $I^1 = \emptyset$, $M^1 = [\,]$

`repetir`

Dados k, I^k e M^k :

Calcular $p_k = s - \left|I^k\right|$

Resolver o sistema

$$\begin{cases} \sum\limits_{i=1}^{p_k} l_i^T A_j l_i + \sum_{i \in I^k} \gamma_i^T A_j m_i = 0, \quad j = 0, 1, \ldots, n, \\ \sum\limits_{i=1}^{p_k} \|l_i\|^2 = 1, \\ l_i^T m_j = 0\ , \quad j \in I^k,\ i = 1, \ldots,\ p_k. \end{cases}$$

`se` o sistema não tem solução, `então` termina e devolve M^k.

`senão`

Dada a solução $\{l_i, i = 1, \ldots, p_k\}$ do sistema, com $l_i \in \mathbb{R}^s$:

Construir o conjunto maximal de $s_k \leq p_k$ vectores, $\{m_1, \ldots, m_{s_k}\} \subset \{l_1, \ldots, l_{p_k}\}$, linearmente independentes dos vectores que constituem a matriz M^k

Fazer $\triangle I^k = \left\{\left|I^k\right| + 1, \ldots, \left|I^k\right| + s_k\right\}$

Actualizar

$M^{k+1} = M^k \cup \left\{m_j,\ j \in \triangle I^k\right\}$

$I^{k+1} = I^k \cup \triangle I^k$

Fazer $k = k + 1$

É evidente que o algoritmo pára após um máximo de s iterações, onde s é a dimensão do espaço das matrizes simétricas do problema que se está a considerar.

B.2 Detalhes da implementação do Algoritmo DIIS em MatLab

A implementação do Algoritmo DIIS, *DIISalgorithm*, não impõe que o utilizador tenha conhecimento profundo do ambiente e da linguagem MatLab. No entanto, o utilizador tem de fazer uma pequena modificação num ficheiro do MatLab, para que tenha disponível a rotina que permite aceder à informação dos problemas de SDP linear em formato SDPA esparso. Esta rotina, bem como o próprio *solver* SDPA-M, está disponível a partir da package SDPA-M, no site

http://grid.r.dendai.ac.jp/sdpa/ .

Esta package contém várias rotinas destinadas a Windows, Solaris ou Linux. Aqui mencionamos o processo de instalação da rotina (para Windows) que necessitamos para a execução correcta do programa.

É então necessário fazer o download do seguinte ficheiro comprimido em formato 'zip':

sdpam.6.2.0.bin.zip .

Depois de descomprimir a package, o utilizador deve copiar os seguintes ficheiros:

- read data.m
- mexsdpa.dll

para a directoria onde pretende trabalhar e onde deverão constar também, o ficheiro do nosso programa, bem como os exemplos em formato SDPA esparso.

Após copiar os ficheiros, deve alterar-se o ficheiro 'startupsav.m' ou 'startup.m' (depende das versões do MatLab), que se encontra na directoria

MATLAB\toolbox\local .

O utilizador apenas tem de adicionar o caminho para a directoria de trabalho que criou anteriormente, da seguinte forma:

no ficheiro 'startupsav.m', acrescenta-se uma linha com

addpath <caminho da directoria de trabalho>.

Suponha-se que o caminho da directoria de trabalho é:

<C:\Users\Isa\Desktop\Isa\matlab exemplos functions\sdpam>;

então a linha que se deve adicionar ao ficheiro é a seguinte:

addpath C:\Users\Isa\Desktop\Isa\matlab exemplos functions\sdpam

Depois desta alteração ao ficheiro 'startupsav.m', o utilizador, quando quiser usar o programa *DIISalgorithm*, apenas tem de abrir o MatLab, seleccionar a pasta onde estão todas as rotinas e ficheiros em formato 'dat-s' necessários à execução do programa, escrever na janela de comandos *DIISalgorithm* e pressionar a tecla *ENTER* para inicializar o programa.

O programa abre uma caixa com todos os exemplos em formato 'dat-s' da directoria de trabalho e é esperado que o utilizador seleccione um deles para o programa começar o procedimento de verificação se se trata de um problema regular ou 'ill-posed'.

A rotina *DIISalgorithm* precisa resolver um sistema de equações não lineares com recurso à função do MatLab *fsolve*. Nesta função tanto a aproximação inicial, como as tolerâncias para as soluções do sistema podem ser alteradas. Numa primeira fase, é perguntado ao utilizador se pretende introduzir uma aproximação inicial para a solução do sistema não linear ou se aceita que o programa gere aleatoriamente uma aproximação. Esta aproximação não tem qualquer relevância nos resultados, serve apenas para determinar uma solução do sistema, uma vez que é usado um Método Quasi-Newtoniano. Quando a rotina principal chamar a função *fsolve* pela primeira vez, é perguntado ao utilizador se pretende alterar os valores das tolerâncias, ou se pretende que se usem as que estão definidas por defeito; as próximas resoluções dos sistemas não lineares usam as opções definidas na primeira fase.

Após o programa concluir a verificação, devolve na janela de comandos as seguintes informações:

- se o sistema de equações não lineares tem ou não solução;
- se o problema de SDP que estamos a testar satisfaz ou não a condição de Slater;
- a dimensão do subespaço $\mathcal{M}$ de índices imóveis;
- a matriz básica M do subespaço de índices imóveis;
- o número de iterações;
- o tempo computacional.

B.3 Exemplo de um problema linear de Programação Semidefinida em formato SDPA esparso

Nesta secção explicamos como se cria um ficheiro em formato SDPA esparso, para consequente leitura no programa *DIISalgorithm*.

O exemplo *polik2* da nossa base de dados de problemas de SDP apresenta-se na forma:

"Example polik2: mDim = 2, nBLOCK = 1, {3}"
2
1
3
{0,1}
0 1 1 1 -1
1 1 2 2 1
2 1 1 1 1
2 1 2 3 1

Na primeira linha escreve-se um comentário, que tem de estar entre aspas.

As próximas 3 linhas correspondem ao número de variáveis do problema, ao número de blocos diagonais que estamos a querer considerar e às dimensões de cada bloco, respectivamente. As dimensões dos blocos podem ser escritas como um vector.

A quinta linha corresponde ao vector dos coeficientes da função objectivo do problema.

As linhas que se seguem correspondem à informação de cada uma das matrizes A_i, com $i = 0, 1, ..., n$. Cada uma destas últimas linhas é composta por

$i\ b\ j\ k\ value$

onde i é o índice da matriz A_i, b é o número do bloco, j e k correspondem à entrada (j, k) da matriz A_i, e $value$ correspondem ao elemento da entrada (j, k) da matriz A_i.

Como consideramos que as matrizes A_i, com $i = 0, 1, ..., n$, são matrizes simétricas, só precizamos guardar a informação da parte triangular superior destas matrizes.

Para criar os ficheiros em formato SDPA esparso em MatLab, abrimos um novo $M-file$ e escrevemos os dados do problema da forma explicada anteriormente. Ao guardar o ficheiro, escrevemos o nome que pretendemos dar ao ficheiro e guardamo-lo com a extensão 'dat-s'; ou seja, o exemplo *polik2* foi guardado como *polik2.dat-s* .

B.4 Código-fonte do Algoritmo DIIS em Linguagem MatLab

B.4.1 Rotina principal

A rotina principal testa se um dado problema de SDP satisfaz ou não a condição de Slater, fazendo uso também de rotinas auxiliares. No fim do procedimento, a rotina *DIISalgorithm* devolve a informação apurada sobre a regularidade do problema.

```
function DIISalgorithm
%
% DIISalgorithm checks if a Linear SDP problem satisfies the Slater condition.
%
% DIISalgorithm attempts to check problems of the form:
%
% min_{X de R^n} C^TX
% s.a A_1*X(1)+A_2*X(2)+...+A_n*X(n)+A_0 >= 0
%
% or
%
% min_{X de R^n} C^TX
% s.a A_1*X(1)+A_2*X(2)+...+A_n*X(n)+A_0 <= 0
%
% where
% A_i, with i=0,1,...,n are symmetric sxs matrices and s>1.
%
% DIISalgorithm implements the DIIS Algorithm, proposed by Kostyukova and
% Tchemisova (2010), to determine a basis of the subspace of immobile
% indices that allows us to check if a Linear SDP problem satisfies the
% Slater condition.
%
% DIISalgorithm checks the problem defined in the file 'problem.dat-s', that
% we've selected from the current directory, where 'dat-s' is the extension
% of the SDPA sparse format file.
% The file 'problem.dat-s', must have the following form:
%
%
%       "problem example
%       3=mDIM
%       1=nBLOCK
%       2=bLOCKsTRUCT
%       {1}
```

```
%       0 1 1 2 1
%       1 1 1 1 1
%       2 1 1 1 1
%
%
% The first line is a comment. The next three lines correspond to the number
% of variables, the number of quadratic blocks and the dimension of the
% blocks, respectively. The fifth line corresponds to the cell vector c in
% the objective function. The next lines correspond to the information of
% each A_i matrices, with i=0,1,...,n. Each of these last lines is
% composed as:
% i b j k value
% where i is the index of A_i, b is the block number, j and k are the
% entry (j,k) in A_i, and value is the value of that entry in A_i.
% We consider that A_i matrices, with i =0,1,...,n, are symmetric, and
% so, we only need to store the upper triangular part of those matrices.
%
%
% Author: Eloísa Macedo
% Time-stamp: 2010-09-26 19:03:46
% E-mail: macedo@ua.pt
%
%
% References:
% [1]Kostyukova O.I. and Tchemisova T.V., "Optimality Criterion without
% Constraint Qualification for Linear Semidefinite Problems", 2010.
% [2]Macedo, E., "Estudo prático de regularidade de problemas de Programação
% Semidefinida", Master Thesis, University of Aveiro, 2010.
% [3]K. Fujisawa, Y. Futakata, M. Kojimay, S. Matsuyama, S. Nakamura, K.
% Nakataz and M. Yamashita, "SDPA-M (SemiDefinite Programming Algorithm in
% MATLAB) User's Manual - Version 6.2.0", Series B: Operations Research
% Department of Mathematical and Computing Sciences, May 2005.
%
%
% This library is a free software; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public
% License as published by the Free Software Foundation.
%
% This library is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU
% Lesser General Public License for more details.
```

```
%
%begin
fprintf(1,'\nSelecting a problem... \n');
prob_base=uigetfile('*.dat-s')
if prob_base==0
  fprintf(1,'\nYou must select a Linear SDP problem in SDPA sparse format \nfrom
...
  ... the current working directory. \n\nType «help DIISalgorithm» for help. \n');
  return
end
fprintf(1,'\nCheck regularity of SDP problems using the DIIS Algorithm \n\n');
global n s nBLOCK bLOCKsTRUCT A conjuntoM
format short
%read data from prob_base, which is in SDPA sparse format
[n,nBLOCK,bLOCKsTRUCT,~,A]=read_data(prob_base); %applies a routine from ...
  ... SDPA-M package to read files in SDPA sparse format: read_data
% check if everything is ok with matrices:
%A{1,1};
%full(A{1,1});
%here:
%n= number of variables: mDim in file
%nBLOCK= number of blocks in A_i, i=0,1,...,n
%bLOCKsTRUCT= dimension of each block (could be a matrix if we have more
%than one quadratic block)
%c= cell vector of objective function's coefficients
%A= A_i matrices in SDPA sparse format
s=sum(abs(bLOCKsTRUCT),1); %the sum of dimensions of bLOCKsTRUCT ...
  ... gives the number of constraints of the problem
for k=1:10, %number of maximum iterations of the DIIS Algorithm
  Q=[]; %auxiliary matrix to concatenate the L vectors; at beginning, Q=empty matrix
  if k==1 %first iteration
    conjuntoI=[]; %I1=empty set
    conjuntoM=[]; %M=empty matrice
    deltaI=[];
    ficheiro=@nlsys1; %calling first form of nonlinear system to solve
  else
    ficheiro=@nlsys2; %calling second form of nonlinear system to solve
  end
  conjuntoI=[conjuntoI deltaI]; %Ik+1=Ik plus deltaI
  p=s-length(conjuntoI); %p_k=number of L vectors; at beginning, p_k=s
  if  p~=0
  if k==1
    %solve the nonlinear system
```

```
        fprintf(1,'Current options in optimset are: ');
        optimset('Display','iter','MaxIter',50000000,'MaxFunEvals',50000000, ...
          ... 'TolX',1e-15,'TolFun',1e-17);
        opt=3;
        while (opt~=1  opt~=2)
          opt=input('Do you want to make changes in optimset? \n (1 - Yes) ...
          ... \n (2 - No) \nand then press ENTER ');
          switch opt
            case 1
              tolx=input('Introduce de value of TolX: ');
              tolfun=input('Introduce de value of TolFun: ');
              options=optimset('Display','iter','MaxIter',50000000,'MaxFunEvals', ...
              ... 50000000, 'TolX',tolx,'TolFun',tolfun);
            case 2
              options=optimset('Display','iter','MaxIter',50000000,'MaxFunEvals', ...
              ... 50000000, 'TolX',1e-15,'TolFun',1e-17);
              fprintf(1,'Using default options...');
            otherwise
              fprintf(1,'\nYou need to introduce a valid option. \n');
          end
        end
      else
        options=optimset('Display','iter','MaxIter',50000000,'MaxFunEvals', ...
          ... 50000000, 'TolX',1e-15,'TolFun',1e-17);
      end
        fprintf(1,'\nYou need to introduce an initial guess for vectors L \nand it must be
...
          ... defined in only one column matrix. \n');
        fprintf(1,'The dimension of that matrix must be: '); comp=s^2; ...
          ... fprintf(' %d \n\n', comp)
        exitflag=0;
        if k==1
          val=4; %false start
          while (exitflag==0)
            while (val~=1 && val~=2)
              val=input('Introduce the appropriate option: \n ...
              ... (1 - if you want introduce yourself the column matrix) \n ...
              ... (2 - if you want that programme generates that column matrix) \n ...
              ... (3 - Exit DIISalgorithm? ) \nand press ENTER    ');
              switch val
              case 1
              L_0=input('Introduce an initial approximation (must be a column matrix):
');
```

```
                tic;
                case 2
                L_0=rand([comp,1]); %randomly generates the inicial approximation L_0
                tic;
                case 3
                fprintf(1,'\nQuitting DIISalgorithm... \n');
                return;
                otherwise
                fprintf(1,'\nYou need to introduce a valid option. \n');
                end
              end
              %check if the matrix intruduced is a column one to proceed
              if ismatrixcol(L_0) && length(L_0)==comp && isnumeric(L_0)
                [L,~,exitflag,output]=fsolve(ficheiro,L_0,options);
              else
                fprintf(1,'You need to introduce a column matrix with length: %d \n',comp);
                L_0=input('Introduce an initial approximation (must be a column matrix):
');
              end
            end
          %for k>1
          else
            %this section does not interact with users
            fprintf(1,'The programme will generate the new column matrix with ...
              ... length: %d \n',comp);
            L_0=rand([comp,1]);
            [L,~,exitflag,output]=fsolve(ficheiro,L_0,options);
            end %end of system of nonlinear equations
          end %end if k=1 or k>1
      %check solution of the system of nonlinear equations
        if exitflag<0 ||p==0
        %the system doesn't have solution: the routine stops
          if exitflag<0
            fprintf(1,'The system of nonlinear equations does not have a solution. \n');
          end
          %just to see       %conjuntoI
          dimM=size(conjuntoM,1);
          if dimM==0
            fprintf(1,'The Linear SDP problem we are checking satisfies the Slater ...
              ... condition. \n');
          else
            fprintf(1,'The Linear SDP problem we are checking does not satisfy the ...
              ... Slater condition. \n');
```

```
    end
    fprintf(1,'The dimension of the subspace of immobile indices is: ');
    dimM
    fprintf(1,'The basic matrix of the subspace of immobile indices is: ');
    conjuntoM
    fprintf(1,'\nNumber of iterations of DIIS algorithm: ');
    fprintf(1,'%d \n',k);
    toc %Elapsed time
    return; %FIM
  else %there is solution! exitflag =0
    L=L'; %solution from fsolve: we have p_k vectors concatenated in the ...
      ...column matrix, L
    L=rounddec(L,3); %round to 3 decimals
    num_elem=s; %number of elements in each L vector or Gamma vector
    if isempty(size(conjuntoM,1))
      fim=length(L); %only L's
    else
      fim=length(L)-size(conjuntoM,1)*num_elem; %when solution from fsolve has ...
      ... Gammas and L's: last elements until length(L) are Gammas
    end
    j=1; %auxiliary counter in the construction of the matrix with each L ...
      ... solution from fsolve
    for i=1:fim, %we have L=[l11...l1i...l1s]
      LL=L(1,j:i);
      controlo=rem(i,num_elem); %control: i/number_of_elem_L =rest. ...
        ... if control=0 =>we introduce L vector in Q
      if controlo==0,
        Q=[Q;LL]; %we've introduced L in Q, and so, the index=j, marks the ...
        ... beginning of other L vector
        j=i+1;
      end %end if
    end %end for
    %on first iteration..
    if k==1,
      Q; %just to see
      %determine whose (L) vectors from Q are linearly independent
      [R,pivot]=rref([Q' zeros(num_elem,1)]) %when R=Identity this means that ...
        ... all vectors from Q are linearly independent
      %conjuntoM
      Q=Q(pivot,:); %change in Q: only linearly independent vectors with indices ...
        ... from pivot matrix; vectors in Q are row vectors
      conjuntoM=Q; %conjuntoM matrix with row vectors
      sk=size(conjuntoM,1);
```

```
    end %end if k=1
    %further iterations..
    if k>1
      %just to see
      conjuntoM; %conjuntoM: each line is a linearly independent vector
      L; %solution from fsolve using csi2
      Q; %this matrix keeps the L row vectors
      clear varargin
      for i=1:size(conjuntoM,1)
        varargin{i}=conjuntoM(i,:);
      end
      sk=0; %auxiliary counter for linearly independent vectors we will introduce
      for q=1:size(Q,1)
        %which vector to test?
        fprintf(1,'\nVector to test linear independence: \n');
        fprintf(1,' %d ',Q(q,:));
        fprintf(1,'\n\n');
        islinindep=linindep(Q(q,:),varargin{:});
        if islinindep==1
        conjuntoM=[conjuntoM; Q(q,:)]; %if current L is linearly independent with ...
        ... all vectors in conjuntoM, we introduce another row with L
        sk=sk+1;
        end %end if
        %just to see
        conjuntoM;
      end  %end for
    end %end if k>1
    deltaI=length(conjuntoI)+1:length(conjuntoI)+sk;
  end  %end if: check if there is solution
end %end programme
```

B.4.2 Rotinas auxiliares

nlsys1

Esta rotina constrói o sistema de equações não lineares a ser resolvido na primeira iteração ($k = 1$). O sistema será resolvido a partir da rotina principal pela função do MatLab *fsolve*.

```
function G=nlsys1(X)
%
% This routine constructs the first nonlinear system (when we only have
```

```
% L's) that we need to solve in the main routine.
%
% Author: Eloísa Macedo
% Time-stamp: 2010-09-26 21:07:01
% E-mail: macedo@ua.pt
%
%
global A s n bLOCKsTRUCT
G=[]; %auxiliary dynamic matrix: sum of L'*Ak*L; at beginning, G is empty
Norma=[]; %auxiliary dynamic matrix: sum of norm^2 of vectors L; at beginning, ...
   ... Norma is empty
for k=1:n+1
   T=[]; %auxiliary matrix in the construction of each line of constraints
   Ak=zeros(s,s);
   r=0; %auxiliary counter
   for p=1:length(bLOCKsTRUCT)
      if bLOCKsTRUCT(p)<0 %the column matrix must be corrected to a diagonal one
         Bk=zeros(abs(bLOCKsTRUCT(p))); %inicialize a square matrix full of zeros
         V=full(A{p,k});
         for i=1:length(V)
            Bk(i,i)=V(i);
         end
      end
      if bLOCKsTRUCT(p)>0
         if isempty(A{p,k})
            Bk=zeros(abs(bLOCKsTRUCT(p)));
         else
            Bk=full(A{p,k});
         end
      end
      Ak(r+1:(r+abs(bLOCKsTRUCT(p))),r+1:(r+abs(bLOCKsTRUCT(p))))=Bk;
      r=r+abs(bLOCKsTRUCT(p));
   end
   Ak;
%equations of the system
% L_0 is a column matrix
% example: how to reach each element from L%%%%%%%
% number of L's =s =3
% number of elements in L =3
% L1=[X(1):X(3)]
% L2=[X(3+1):X(2*3)]
% L3=[X(6+1):X(3*3)]
%
```

```
    r=0;
    %for A_k:
    for j=s:s:s^2, %s^2= index of the last element in L, solution from fsolve
      L=X(r+1:j);
      T=[T;L'*Ak*L]; %this matrix keeps every product Li'*Ak*Li, in each line i
      r=r+s;
    end
    G=[G;sum(T,1)]; %this matrix sum all the above products
    %now, changing matrix: Aj=A_k+1
  end
  r=0; %new auxiliary counter
  for i=s:s:s^2, %
    L=X(r+1:i); %
    Norma=[Norma;norm(L)^2]; %the same as before, but now in terms of ...
      ... keeping norm^2 of each vector Li
    r=r+s; %
  end %
  G=[G; sum(Norma,1)-1]; %augmenting one row in G with: (sum_of_all norm^2)-1
```

nlsys2

A rotina *nlsys2* constrói os sistemas não lineares resultantes do procedimento. Esta rotina é chamada a partir da iteração 2. De forma análoga ao que acontece com a rotina *nlsys1*, o sistema resultante será resolvido a partir da rotina principal.

```
function G=nlsys2(X)
%
% This routine constructs the second nonlinear system (when we have L's and Gammas)
% that we need to solve in the main routine.
%
% Author: Eloísa Macedo
% Time-stamp: 2010-09-26 21:21:54
% E-mail: macedo@ua.pt
%
%
global A s n bLOCKsTRUCT conjuntoM
conjuntoM;
G=[]; %auxiliary dynamic matrix: sum of L'*Ak*L; at beginning, G is empty
Norma=[]; %auxiliary dynamic matrix: sum of norm^2 of vectors L; at beginning, ...
  ... Norma is empty
Lm=[]; %auxiliary dynamic matrix for Li'*mj with mj a vector from conjuntoM
Gam=[]; %auxiliary dynamic matrix: sum of Gamai'*Ak*mj
```

```
mes=size(conjuntoM,1);
c=mes*s; %auxiliary counter: s^2-c=indices reserved for all 'mes' Gamma vectors
for k=1:n+1
   T=[]; %auxiliary matrix in the construction of each line of constraints
   r=0; %auxiliary counter
   Ak=zeros(s,s);
   Bk=[];
   for p=1:length(bLOCKsTRUCT)
      if bLOCKsTRUCT(p)<0 %the column matrix must be corrected to a diagonal one
         Bk=zeros(abs(bLOCKsTRUCT(p))); %inicialize a square matrix full of zeros
         V=full(A{p,k});
         for i=1:length(V)
            Bk(i,i)=V(i);
         end
      end
      if bLOCKsTRUCT(p)>0
         if isempty(A{p,k})
            Bk=zeros(abs(bLOCKsTRUCT(p)));
         else
            Bk=full(A{p,k});
         end
      end
      Ak(r+1:(r+abs(bLOCKsTRUCT(p))),r+1:(r+abs(bLOCKsTRUCT(p))))=Bk;
      r=r+abs(bLOCKsTRUCT(p));
   end
   Ak;
   r=0;
   %for A_k:
   for j=s:s:(s^2-c), %s^2= index of the last element in L, solution from fsolve
      L=X(r+1:j);
      T=[T;L'*Ak*L]; %this matrix keeps every product Li'*Ak*Li, in each line i
      r=r+s;
   end
   for j=(s^2-c+s):s:s^2,
      L=X(r+1:j);
      for h=1:mes,
         Gam=[Gam;L'*Ak*conjuntoM(h,:)']; %this matrix keeps every ...
            ... product Gami'*Ak*mi
      end
      r=r+s;
   end
   G=[G;sum(T,1)+sum(Gam,1)]; %this matrix sum all the above products:...
      ... introduces k rows in G
```

```
    %now, changing matrix: Aj=A_k+1
end
r=0; %new auxiliary counter
for i=s:s:(s^2-c), %
    L=X(r+1:i); %
    Norma=[Norma;norm(L)^2]; %the same as before, but now in terms of...
        ... keeping norm^2 of each vector Li
    r=r+s; %
end %
G=[G; sum(Norma,1)-1]; %augmenting one row in G with: (sum_of_all norm^2)-1
r=0; %new auxiliary counter
for i=s:s:(s^2-c), %
    L=X(r+1:i); %
    for h=1:mes, %
        Lm=[Lm;L'*conjuntoM(h,:)']; %the same as before, but now in terms of ...
            ... keeping Li'*mj: introduces one row in G
        G=[G; Lm]; %
    end %
    r=r+s; %
end  %
G ; %final G matrix
```

linindep

Esta rotina verifica se um vector é linearmente independente com todos os vectores constituintes de um conjunto.

```
function islinindep=linindep(v, varargin)
%
% This routine checks if the row vector v is linearly independent of the
% set of row vectors defined until here.
% if they are linearly independent: islinindep=1
% otherwise: islinindep=2
%
% Author: Eloísa Macedo
% Time-stamp: 2010-09-26 19:03:46
% E-mail: macedo@ua.pt
%
%
P=[]; %dynamic matrix: starts empty
n=length(varargin); %number of arguments: arguments are row vectors
for i=1:n
```

```
u=varargin{i};
u=u'; %row vector u (argument in the i-th position) changes to column vector
P=[P u(:)]; %P is augmented with column u: argument vectors are now in columns
end
v=v'; %vector to test linear independence with the others in arguments
v=v(:);
if rank(P)==rank([P v]) %rank [P v]>rank [P] => v is linearly independent ...
   ... with all the other vectors in P
islinindep=2;
fprintf(1,' The vector is linear combination of the current vectors.\n');
else
islinindep=1;
fprintf(1,' The vectors are linearly independent.\n');
end
```

ismatrixcol

Esta rotina verifica se a matriz dada como argumento é, ou não, uma matriz coluna.

```
function ismc=ismatrixcol(u)
%
% This routine checks if the argument is a column matrix.
% if so: ismc=1
% otherwise: ismc=2
%
% Author: Eloísa Macedo
% Time-stamp: 2010-09-26 19:03:46
% E-mail: macedo@ua.pt
%
%
ismc=(ndims(u)== 2) & (min(size(u,2))== 1);
if ismc==1
   disp('It is a column matrix.');
else
   disp('It is not a column matrix.');
end
end
```

rounddec

Esta função é de livre acesso na internet (ver Acklam [71]) e permite arredondar um argumento a um determinado número de casas decimais.

```
function y = rounddec(x, n)
%ROUNDDEC Round to a specified number of decimals.
%
% Y = ROUNDDEC(X, N) rounds the elements of X to N decimals.
%
% For instance, rounddec(10*sqrt(2) + i*pi/10, 4) returns
% 14.1421 + 0.3142i
%
% See also: ROUND, FIX, FLOOR, CEIL, ROUNDDIG, TRUNCDEC, TRUNCDIG.
% Author: Peter J. Acklam
% Time-stamp: 2004-09-22 20:06:46 +0200
% E-mail: pjacklam@online.no
% URL: http://home.online.no/~pjacklam
% Check number of input arguments.
error(nargchk(2, 2, nargin));
% Quick exit if either argument is empty.
if isempty(x) ||isempty(n)
   y = [];
   return
end
% Get size of input arguments.
size_x = size(x);
size_n = size(n);
scalar_x = all(size_x == 1); % True if x is a scalar.
scalar_n = all(size_n == 1); % True if n is a scalar.
% Check size of input arguments.
if ~scalar_x && ~scalar_n && ~isequal(size_x, size_n)
   error(['When both arguments are non-scalars they must have the same size']);
end
f = 10.^n;
y = round(x .* f) ./ f; %round if a function from MatLab
end
```

read_data.m

Esta rotina faz parte da package SDPA-M. De qualquer forma, transcrevemos a rotina *read_data.m*, que permite ter acesso aos dados do problema de SDP no formato 'dat-s'.

```
function [mDIM,nBLOCK,bLOCKsTRUCT,c,F]=read_data(filename);
%
% Read a problem in SDPA sparse format.
%
% [mDIM,nBLOCK,bLOCKsTRUCT,c,F] = read_data(fname)
%
% INPUT
% - filename: string; filename of the SDP data with SDPA foramt.
%
% OUTPUT
% - mDIM : integer; number of primal variables
% - nBLOCK : integer; number of blocks of F
% - bLOCKsTRUCT: vector; represents the block structure of F
% - c : vector; coefficient vector
% - F : cell array; coefficient matrices
%
% This file is a component of SDPA
% Copyright (C) 2004 SDPA Project
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 2 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
%
% SDPA-M: Revision: 6.2
% Id: read_data.m,v 6.2 2005-05-28 02:36:40 drophead Exp
% check the validity of the arguments
if ( nargin ~= 1 | (nargin == 1ïisstr(filename)))
error('input argument must be a filename');
end
% identify whether a file is sparse format or not.
bsparse=0;
len=length(filename);
```

```
if len >= 2
   str=filename(end-1:end);
   if strncmp(str,'-s',2)
      bsparse=1;
   end
end
fid=fopen(filename,'r');
if fid == -1
   error(sprintf('Cannot open %s',filename));
end
% skip comment and after it, read a number of decision variables (mDIM)
while 1
   str=fgetl(fid);
   if ( str(1)~='*'  str(1) ~='"' )
      mDIM=sscanf(str,'%d',1);
      break;
   end
end
% disp(sprintf('mDIM=%d',mDIM));
% read a number of blocks (nBLOCK)
nBLOCK=fscanf(fid,'%d',1);
% disp(sprintf('nBLOCK=%d',nBLOCK));
% read each size of blocks (bLOCKsTRUCT)
bLOCKsTRUCT=zeros(nBLOCK,1);
for idx=1:nBLOCK
   bLOCKsTRUCT(idx)=fscanf(fid,'%*[^0-9+-]%d',1);
   if bLOCKsTRUCT(idx) == 1
      bLOCKsTRUCT(idx) = -1;
   end
end
% read cost vector (c)
c=zeros(mDIM,1);
for idx=1:mDIM
   c(idx)=fscanf(fid,'%*[^0-9+-]%lg',1);
end
% read coefficient matrices (F)
F=cell(nBLOCK,mDIM+1);
if bsparse
   % sparse format case
   while 1
      [k,cnt]=fscanf(fid,'%*[^0-9+-]%d',1);
      [l,cnt]=fscanf(fid,'%*[^0-9+-]%d',1);
      [i,cnt]=fscanf(fid,'%*[^0-9+-]%d',1);
```

```
        [j,cnt]=fscanf(fid,'%*[^0-9+-]%d',1);
        [value,cnt]=fscanf(fid,'%*[^0-9+-]%lg',1);
        if cnt
          if isempty(Fl,k+1)
          size=abs(bLOCKsTRUCT(l));
          if bLOCKsTRUCT(l) ¡ 0
            Fl,k+1=sparse(zeros(size,1));
          else
            Fl,k+1=sparse(zeros(size));
          end
        end
        if bLOCKsTRUCT(l) ¡ 0
          Fl,k+1(i)=value;
        else
          if i ¡ j
            Fl,k+1(i,j)=value;
            Fl,k+1(j,i)=value;
            elseif i == j
              Fl,k+1(i,j)=value;
            end
          end
          else
          break;
        end
      end
    else
      % dense format case
      for k=1:mDIM+1
        for l=1:nBLOCK
          size=abs(bLOCKsTRUCT(l));
          if bLOCKsTRUCT(l) ¿ 0
            Fl,k=zeros(size);
            for i=1:size
              for j=1:size
                [value,cnt]=fscanf(fid,'%*[^0-9+-]%lg',1);
                if cnt
                Fl,k(i,j)=value;
                else
                error(sprintf('Failed to read an element at %d %d %d %d'), k-1,l,i,j);
                end
              end
            end
          else
```

```
            Fl,k=zeros(size,1);
            for i=1:size
               [value,cnt]=fscanf(fid,'%*[^0 - 9 + -]%lg', 1);
               if cnt
                  Fl,k(i)=value;
               else
                  error(sprintf('Failed to read an element at %d %d %d %d'), k-1,l,i,i);
               end
            end
         end
      end
   end
end
fclose(fid);
% End of File
```

Apêndice C

Transformação de problemas de LP para SDP

Nesta secção vamos incluir os exemplos de problemas de LP transformados em problemas de SDP usados para o teste do programa desenvolvido, *DIISalgorithm*.

C.1 Exemplos usados no teste do programa *DIISalgorithm*

LPtoSDP1

Considere-se o problema de LP:

$$\begin{aligned} & \min 2x_1 + 2x_2 \\ \text{s. a } & -2x_1 + x_2 \geq 1 \\ & x_1 - 2x_2 \geq 1. \end{aligned} \tag{C.1}$$

Uma vez que um problema de LP pode ser considerado um caso particular de SDP (supondo que as matrizes A_i são diagonais), o problema (C.1) pode ser escrito como um problema de SDP, fazendo

$$A_0 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, A_1 = \begin{bmatrix} -2 & 0 \\ 0 & 1 \end{bmatrix} \text{ e } A_2 = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix}$$

e obtemos o problema

$$\begin{aligned} & \min 2x_1 + 2x_2 \\ \text{s. a } & \begin{bmatrix} -2 & 0 \\ 0 & 1 \end{bmatrix} x_1 + \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix} x_2 + \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \succeq 0. \end{aligned}$$

Nota 4 *O programa DIISalgorithm aceita problemas semidefinidos negativos ou positivos, uma vez que os dados dos problemas são usados para construir um sistema de equações não lineares, e portanto, uma mudança de sinal transforma o sistema num outro equivalente, não alterando as soluções.*

LPtoSDP2

Considere-se o problema de LP:

$$\begin{aligned} & \min -x_1 + x_2 \\ \text{s. a} \quad & x_1 - x_2 \geq 1 \\ & 2x_1 - x_2 \geq -4 \\ & -x_1 + 2x_2 - 2 \geq 0. \end{aligned} \tag{C.2}$$

A forma equivalente de (C.2) como problema de SDP é

$$\begin{aligned} & \min -x_1 + x_2 \\ \text{s. a } & A_1x_1 + A_2x_2 + A_0 \succeq 0, \end{aligned}$$

onde

$$A_0 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -2 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \text{ e } A_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

LPtoSDP3

Considere-se o problema de LP:

$$\begin{aligned} & \min x_1 - 2x_2 \\ \text{s. a} \quad & x_1 + x_2 - 8 \geq 0 \\ & -2x_1 + x_2 \geq -2 \\ & -x_1 + 2x_2 - 2 \geq 6. \end{aligned} \tag{C.3}$$

A forma equivalente de (C.3) como problema de SDP é

$$\begin{aligned} & \min x_1 - x_2 \\ \text{s. a } & A_1x_1 + A_2x_2 + A_0 \succeq 0, \end{aligned}$$

onde

$$A_0 = \begin{bmatrix} -8 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -8 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \text{ e } A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

LPtoSDP4

Considere-se o problema de LP:

$$\begin{aligned} & \min 5x_1 + 2x_2 + 4x_3 \\ \text{s. a} \quad & 3x_1 + x_2 + 2x_3 - 5 \geq 0 \\ & 6x_1 + 3x_2 + 5x_3 - 10 \geq 0. \end{aligned} \tag{C.4}$$

A forma equivalente de (C.4) como problema de SDP é

$$\begin{aligned} & \min 5x_1 + 2x_2 + 4x_3 \\ \text{s. a } & A_1x_1 + A_2x_2 + A_3x_3 + A_0 \succeq 0, \end{aligned}$$

onde

$$A_0 = \begin{bmatrix} -5 & 0 \\ 0 & -10 \end{bmatrix}, A_1 = \begin{bmatrix} 3 & 0 \\ 0 & 6 \end{bmatrix}, A_2 = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix} \text{ e } A_3 = \begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}$$

LPtoSDP5

Considere-se o problema de LP:

$$\begin{aligned} & \min x_1 + x_2 + x_3 \\ \text{s. a} \quad & x_1 - x_2 - 5 \geq 0 \\ & 2x_1 - 3x_2 + x_3 \geq 12 \\ & 2x_1 - 52x_2 + 6x_3 \geq 5. \end{aligned} \tag{C.5}$$

A forma equivalente de (C.5) como problema de SDP é

$$\begin{aligned} & \min x_1 + x_2 + x_3 \\ \text{s. a } & A_1x_1 + A_2x_2 + A_3x_3 + A_0 \succeq 0, \end{aligned}$$

onde

$$A_0 = \begin{bmatrix} -5 & 0 & 0 \\ 0 & -12 & 0 \\ 0 & 0 & -5 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, A_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -52 \end{bmatrix} \text{ e}$$

$$A_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 6 \end{bmatrix}.$$

LPtoSDP6

Considere-se o problema de LP:

$$\begin{aligned} & \min -x_1 + x_2 \\ \text{s. a} \quad & -x_1 - x_2 - 1 \geq 0 \\ & x_1 \geq 0 \\ & x_2 \geq 0. \end{aligned} \tag{C.6}$$

A forma equivalente de (C.6) como problema de SDP é

$$\begin{aligned} & \min -x_1 + x_2 \\ \text{s. a } & A_1 x_1 + A_2 x_2 + A_0 \succeq 0, \end{aligned}$$

onde

$$A_0 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \; A_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ e } A_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$